Placement Documentation

Release 6.0.1.dev2

OpenStack Foundation

May 10, 2022

CONTENTS

The placement API service was introduced in the 14.0.0 Newton release within the nova repository and extracted to the placement repository in the 19.0.0 Stein release. This is a REST API stack and data model used to track resource provider inventories and usages, along with different classes of resources. For example, a resource provider can be a compute node, a shared storage pool, or an IP allocation pool. The placement service tracks the inventory and usage of each provider. For example, an instance created on a compute node may be a consumer of resources such as RAM and CPU from a compute node resource provider, disk from an external shared storage pool resource provider and IP addresses from an external IP pool resource provider.

The types of resources consumed are tracked as **classes**. The service provides a set of standard resource classes (for example `DISK_GB`, `MEMORY_MB`, and `VCPU`) and provides the ability to define custom resource classes as needed.

Each resource provider may also have a set of traits which describe qualitative aspects of the resource provider. Traits describe an aspect of a resource provider that cannot itself be consumed but a workload may wish to specify. For example, available disk may be solid state drives (SSD).

CHAPTER

ONE

USAGES

1.1 Placement Usage

1.1.1 Tracking Resources

The placement service enables other projects to track their own resources. Those projects can register/delete their own resources to/from placement via the placement HTTP API.

The placement service originated in the Nova project. As a result much of the functionality in placement was driven by nova's requirements. However, that functionality was designed to be sufficiently generic to be used by any service that needs to manage the selection and consumption of resources.

How Nova Uses Placement

Two processes, `nova-compute` and `nova-scheduler`, host most of nova's interaction with placement.

The nova resource tracker in `nova-compute` is responsible for creating the resource provider record corresponding to the compute host on which the resource tracker runs, setting the inventory that describes the quantitative resources that are available for workloads to consume (e.g., `VCPU`), and setting the traits that describe qualitative aspects of the resources (e.g., `STORAGE_DISK_SSD`).

If other projects -- for example, Neutron or Cyborg -- wish to manage resources on a compute host, they should create resource providers as children of the compute host provider and register their own managed resources as inventory on those child providers. For more information, see the *Modeling with Provider Trees*.

The `nova-scheduler` is responsible for selecting a set of suitable destination hosts for a workload. It begins by formulating a request to placement for a list of allocation candidates. That request expresses quantitative and qualitative requirements, membership in aggregates, and in more complex cases, the topology of related resources. That list is reduced and ordered by filters and weighers within the scheduler process. An allocation is made against a resource provider representing a destination, consuming a portion of the inventory set by the resource tracker.

Modeling with Provider Trees

Overview

Placement supports modeling a hierarchical relationship between different resource providers. While a parent provider can have multiple child providers, a child provider can belong to only one parent provider. Therefore, the whole architecture can be considered as a "tree" structure, and the resource provider on top of the "tree" is called a "root provider". (See the Nested Resource Providers spec for details.)

Modeling the relationship is done by specifying a parent provider via the POST /resource_providers operation when creating a resource provider.

Note: If the parent provider hasn't been set, you can also parent a resource provider after the creation via the PUT /resource_providers/{uuid} operation. But re-parenting a resource provider is not supported.

The resource providers in a tree -- and sharing providers as described in the next section -- can be returned in a single allocation request in the response of the GET /allocation_candidates operation. This means that the placement service looks up a resource provider tree in which resource providers can *collectively* contain all of the requested resources.

This document describes some case studies to explain how sharing providers, aggregates, and traits work if provider trees are involved in the GET /allocation_candidates operation.

Sharing Resource Providers

Resources on sharing resource providers can be shared by multiple resource provider trees. This means that a sharing provider can be in one allocation request with resource providers from a different tree in the response of the GET /allocation_candidates operation. As an example, this may be used for shared storage that is connected to multiple compute hosts.

Note: Technically, a resource provider with the `MISC_SHARES_VIA_AGGREGATE` trait becomes a sharing resource provider and the resources on it are shared by other resource providers in the same aggregate.

For example, let's say we have the following environment:

```
+-------------------------------+   +-------------------------------+
| Sharing Storage (SS1)         |   | Sharing Storage (SS2)         |
|  resources:                   |   |  resources:                   |
|      DISK_GB: 1000            |   |      DISK_GB: 1000            |
|  aggregate: [aggA]            |   |  aggregate: []                |
|  trait:                       |   |  trait:                       |
|   [MISC_SHARES_VIA_AGGREGATE] |   |   [MISC_SHARES_VIA_AGGREGATE] |
+---------------+---------------+   +-------------------------------+
                | Shared via aggA
    +-----------+-----------+           +-----------------------+
    | Compute Node (CN1)    |           | Compute Node (CN2)    |
    |   resources:          |           |   resources:          |
    |      VCPU: 8          |           |      VCPU: 8          |
```

(continues on next page)

(continued from previous page)

```
    |      MEMORY_MB: 1024  |           |      MEMORY_MB: 1024  |
    |      DISK_GB: 1000    |           |      DISK_GB: 1000    |
    |   aggregate: [aggA]   |           |   aggregate: []       |
    |   trait: []           |           |   trait: []           |
    +-----------------------+           +-----------------------+
```

Assuming no allocations have yet been made against any of the resource providers, the request:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500
```

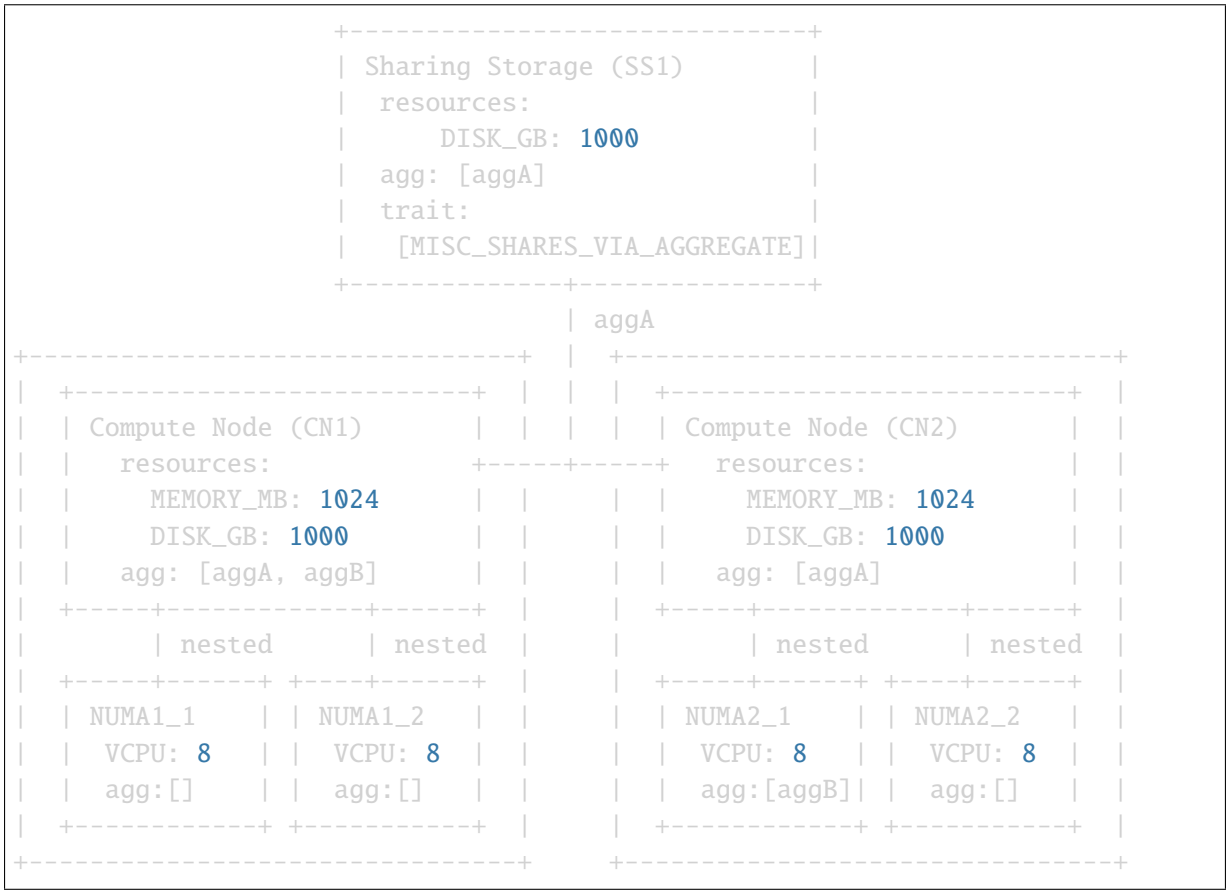
would return three combinations as the allocation candidates.

1. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`)
2. `CN2` (`VCPU`, `MEMORY_MB`, `DISK_GB`)
3. `CN1` (`VCPU`, `MEMORY_MB`) + `SS1` (`DISK_GB`)

`SS2` is also a sharing provider, but not in the allocation candidates because it can't satisfy the resource itself and it isn't in any aggregate, so it is not shared by any resource providers.

When a provider tree structure is present, sharing providers are shared by the whole tree if one of the resource providers from the tree is connected to the sharing provider via an aggregate.

For example, let's say we have the following environment where NUMA resource providers are child providers of the compute host resource providers:



Assuming no allocations have yet been made against any of the resource providers, the request:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500
```

would return eight combinations as the allocation candidates.

1. NUMA1_1 (VCPU) + CN1 (MEMORY_MB, DISK_GB)
2. NUMA1_2 (VCPU) + CN1 (MEMORY_MB, DISK_GB)
3. NUMA2_1 (VCPU) + CN2 (MEMORY_MB, DISK_GB)
4. NUMA2_2 (VCPU) + CN2 (MEMORY_MB, DISK_GB)
5. NUMA1_1 (VCPU) + CN1 (MEMORY_MB) + SS1 (DISK_GB)
6. NUMA1_2 (VCPU) + CN1 (MEMORY_MB) + SS1 (DISK_GB)
7. NUMA2_1 (VCPU) + CN2 (MEMORY_MB) + SS1 (DISK_GB)
8. NUMA2_2 (VCPU) + CN2 (MEMORY_MB) + SS1 (DISK_GB)

Note that NUMA1_1 and SS1, for example, are not in the same aggregate, but they can be in one allocation request since the tree of CN1 is connected to SS1 via aggregate A on CN1.

Filtering Aggregates

What differs between the CN1 and CN2 in the example above emerges when you specify the aggregate explicitly in the GET /allocation_candidates operation with the member_of query parameter. The member_of query parameter accepts aggregate uuids and filters candidates to the resource providers in the given aggregate. See the Filtering by Aggregate Membership spec for details.

Note that the GET /allocation_candidates operation assumes that "an aggregate on a root provider spans the whole tree, while an aggregate on a non-root provider does NOT span the whole tree."

For example, in the environment above, the request:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500&member_
↪of=<aggA uuid>
```

would return eight candidates,

1. NUMA1_1 (VCPU) + CN1 (MEMORY_MB, DISK_GB)
2. NUMA1_2 (VCPU) + CN1 (MEMORY_MB, DISK_GB)
3. NUMA2_1 (VCPU) + CN2 (MEMORY_MB, DISK_GB)
4. NUMA2_2 (VCPU) + CN2 (MEMORY_MB, DISK_GB)
5. NUMA1_1 (VCPU) + CN1 (MEMORY_MB) + SS1 (DISK_GB)
6. NUMA1_2 (VCPU) + CN1 (MEMORY_MB) + SS1 (DISK_GB)
7. NUMA2_1 (VCPU) + CN2 (MEMORY_MB) + SS1 (DISK_GB)
8. NUMA2_2 (VCPU) + CN2 (MEMORY_MB) + SS1 (DISK_GB)

This is because aggregate A is on the root providers, CN1 and CN2, so the API assumes the child providers NUMA1_1, NUMA1_2, NUMA2_1 and NUMA2_2 are also in the aggregate A.

Specifying aggregate B:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500&member_
↪of=<aggB uuid>
```

would return two candidates.

1. `NUMA1_1` (`VCPU`) + `CN1` (`MEMORY_MB`, `DISK_GB`)
2. `NUMA1_2` (`VCPU`) + `CN1` (`MEMORY_MB`, `DISK_GB`)

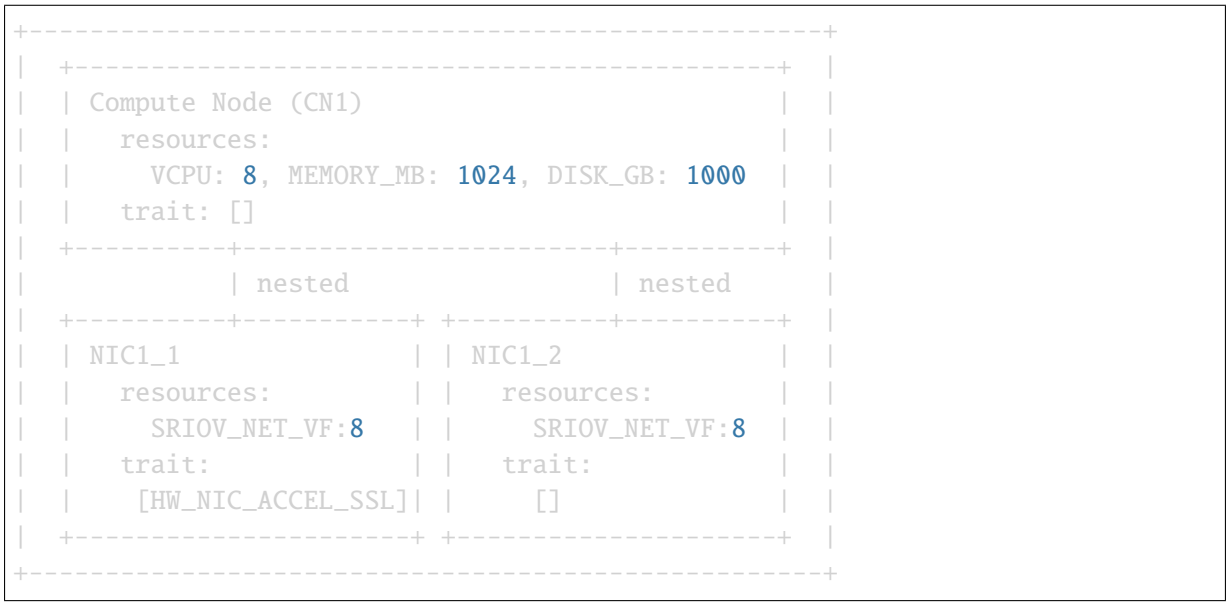
This is because `SS1` is not in aggregate B, and because aggregate B on `NUMA2_1` doesn't span the whole tree since the `NUMA2_1` resource provider isn't a root resource provider.

Filtering by Traits

Traits are not only used to indicate sharing providers. They are used to denote capabilities of resource providers. (See The Traits API spec for details.)

Traits can be requested explicitly in the GET /allocation_candidates operation with the `required` query parameter, but traits on resource providers never span other resource providers. If a trait is requested, one of the resource providers that appears in the allocation candidate should have the trait regardless of sharing or nested providers. See the Request Traits spec for details. The `required` query parameter also supports negative expression, via the `!` prefix, for forbidden traits. If a forbidden trait is specified, none of the resource providers that appear in the allocation candidate may have that trait. See the Forbidden Traits spec for details.

For example, let's say we have the following environment:



Assuming no allocations have yet been made against any of the resource providers, the request:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500,SRIOV_
↪NET_VF:2
                          &required=HW_NIC_ACCEL_SSL
```

would return only `NIC1_1` for `SRIOV_NET_VF`. As a result, we get one candidate.

1. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_1` (`SRIOV_NET_VF`)

In contrast, for forbidden traits:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500,SRIOV_
↪NET_VF:2
                          &required=!HW_NIC_ACCEL_SSL
```

would exclude `NIC1_1` for `SRIOV_NET_VF`.

1. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_2` (`SRIOV_NET_VF`)

If the trait is not in the `required` parameter, that trait will simply be ignored in the GET /allocation_candidates operation.

For example:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500,SRIOV_
↪NET_VF:2
```

would return two candidates.

1. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_1` (`SRIOV_NET_VF`)
2. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_2` (`SRIOV_NET_VF`)

Granular Resource Requests

If you want to get the same kind of resources from multiple resource providers at once, or if you require a provider of a particular requested resource class to have a specific trait or aggregate membership, you can use the Granular Resource Request feature.

This feature is enabled by numbering the `resources`, `member_of` and `required` query parameters respectively.

For example, in the environment above, the request:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500
                          &resources1=SRIOV_NET_VF:1&required1=HW_NIC_ACCEL_
↪SSL
                          &resources2=SRIOV_NET_VF:1
                          &group_policy=isolate
```

would return one candidate where two providers serve `SRIOV_NET_VF` resource.

1. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_1` (`SRIOV_NET_VF:1`) + `NIC1_2` (`SRIOV_NET_VF:1`)

The `group_policy=isolate` ensures that the one resource is from a provider with the `HW_NIC_ACCEL_SSL` trait and the other is from *another* provider with no trait constraints.

If the `group_policy` is set to `none`, it allows multiple granular requests to be served by one provider. Namely:

```
GET /allocation_candidates?resources=VCPU:1,MEMORY_MB:512,DISK_GB:500
                          &resources1=SRIOV_NET_VF:1&required1=HW_NIC_ACCEL_
↪SSL
                          &resources2=SRIOV_NET_VF:1
                          &group_policy=none
```

would return two candidates.

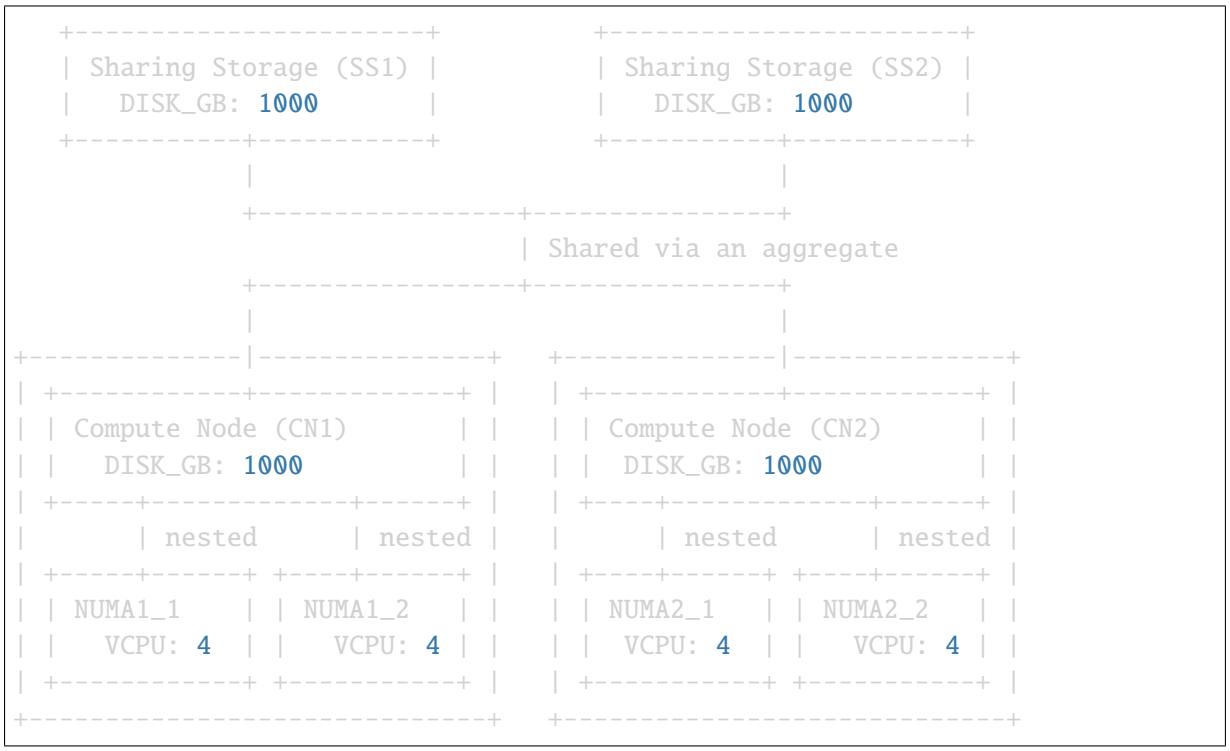
1. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_1` (`SRIOV_NET_VF:1`) + `NIC1_2` (`SRIOV_NET_VF:1`)
2. `CN1` (`VCPU`, `MEMORY_MB`, `DISK_GB`) + `NIC1_1` (`SRIOV_NET_VF:2`)

This is because `NIC1_1` satisfies both request 1 (with `HW_NIC_ACCEL_SSL` trait) and request 2 (with no trait constraints).

Note that if `member_of<N>` is specified in granular requests, the API doesn't assume that "an aggregate on a root provider spans the whole tree." It just sees whether the specified aggregate is directly associated with the resource provider when looking up the candidates.

Filtering by Tree

If you want to filter the result by a specific provider tree, use the Filter Allocation Candidates by Provider Tree feature with the `in_tree` query parameter. For example, let's say we have the following environment:



The request:

```
GET /allocation_candidates?resources=VCPU:1,DISK_GB:50&in_tree=<CN1 uuid>
```

will filter out candidates by `CN1` and return 2 combinations of allocation candidates.

1. `NUMA1_1` (`VCPU`) + `CN1` (`DISK_GB`)
2. `NUMA1_2` (`VCPU`) + `CN1` (`DISK_GB`)

The specified tree can be a non-root provider. The request:

```
GET /allocation_candidates?resources=VCPU:1,DISK_GB:50&in_tree=<NUMA1_1 uuid>
```

will return the same result being aware of resource providers in the same tree with NUMA1_1 resource provider.

1. NUMA1_1 (VCPU) + CN1 (DISK_GB)
2. NUMA1_2 (VCPU) + CN1 (DISK_GB)

Note: We don't exclude NUMA1_2 in the case above. That kind of feature is proposed separately and in progress. See the Support subtree filter specification for details.

The suffixed syntax in_tree<$S> (where $S is a number in microversions 1.25-1.32 and [a-zA-Z0-9_-]{1,64} from 1.33) is also supported according to *Granular Resource Requests*. This restricts providers satisfying the suffixed granular request group to the tree of the specified provider.

For example, in the environment above, when you want to have VCPU from CN1 and DISK_GB from wherever, the request may look like:

```
GET /allocation_candidates?resources=VCPU:1&in_tree=<CN1 uuid>
                          &resources1=DISK_GB:10
```

which will return the sharing providers as well as the local disk.

1. NUMA1_1 (VCPU) + CN1 (DISK_GB)
2. NUMA1_2 (VCPU) + CN1 (DISK_GB)
3. NUMA1_1 (VCPU) + SS1 (DISK_GB)
4. NUMA1_2 (VCPU) + SS1 (DISK_GB)
5. NUMA1_1 (VCPU) + SS2 (DISK_GB)
6. NUMA1_2 (VCPU) + SS2 (DISK_GB)

This is because the unsuffixed in_tree is applied to only the unsuffixed resource of VCPU, and not applied to the suffixed resource, DISK_GB.

When you want to have VCPU from wherever and DISK_GB from SS1, the request may look like:

```
GET /allocation_candidates?resources=VCPU:1
                          &resources1=DISK_GB:10&in_tree1=<SS1 uuid>
```

which will stick to the first sharing provider for DISK_GB.

1. NUMA1_1 (VCPU) + SS1 (DISK_GB)
2. NUMA1_2 (VCPU) + SS1 (DISK_GB)
3. NUMA2_1 (VCPU) + SS1 (DISK_GB)
4. NUMA2_2 (VCPU) + SS1 (DISK_GB)

When you want to have VCPU from CN1 and DISK_GB from SS1, the request may look like:

```
GET /allocation_candidates?resources1=VCPU:1&in_tree1=<CN1 uuid>
                          &resources2=DISK_GB:10&in_tree2=<SS1 uuid>
                          &group_policy=isolate
```

which will return only 2 candidates.

1. NUMA1_1 (VCPU) + SS1 (DISK_GB)
2. NUMA1_2 (VCPU) + SS1 (DISK_GB)

Filtering by Root Provider Traits

When traits are associated with a particular resource, the provider tree should be constructed such that the traits are associated with the provider possessing the inventory of that resource. For example, trait HW_CPU_X86_AVX2 is a trait associated with the VCPU resource, so it should be placed on the resource provider with VCPU inventory, wherever that provider is positioned in the tree structure. (A NUMA-aware host may model VCPU inventory in a child provider, whereas a non-NUMA-aware host may model it in the root provider.)

On the other hand, some traits are associated not with a resource, but with the provider itself. For example, a compute host may be capable of COMPUTE_VOLUME_MULTI_ATTACH, or be associated with a CUSTOM_WINDOWS_LICENSE_POOL. In this case it is recommended that the root resource provider be used to represent the concept of the "compute host"; so these kinds of traits should always be placed on the root resource provider.

The following environment illustrates the above concepts:

```
+--------------------------------+ +-------------------------------------
↪--+
|+------------------------------+| |    +------------------------------+   ␣
↪  |
|| Compute Node (NON_NUMA_CN)   || |    | Compute Node (NUMA_CN)       |   ␣
↪  |
||  VCPU: 8,                    || |    |  DISK_GB: 1000               |   ␣
↪  |
||  MEMORY_MB: 1024             || |    | traits:                      |   ␣
↪  |
||  DISK_GB: 1000               || |    |  STORAGE_DISK_SSD,           |   ␣
↪  |
|| traits:                     || |    |  COMPUTE_VOLUME_MULTI_ATTACH |   ␣
↪  |
||  HW_CPU_X86_AVX2,            || |    +-------+------------+---------+   ␣
↪  |
||  STORAGE_DISK_SSD,           || |     nested |            | nested      ␣
↪  |
||  COMPUTE_VOLUME_MULTI_ATTACH, || |+----------+-------+ +---+--------------
↪-+|
||  CUSTOM_WINDOWS_LICENSE_POOL  || || NUMA1           | | NUMA2          ␣
↪ ||
|+------------------------------+| ||  VCPU: 4         | |  VCPU: 4       ␣
↪ ||
+--------------------------------+ ||  MEMORY_MB: 1024  | |  MEMORY_MB: 1024␣
↪ ||
                                 ||                  | | traits:        ␣
↪ ||
                                 ||                  | |  HW_CPU_X86_AVX2␣
↪ ||
```

(continues on next page)

(continued from previous page)

```
                                    |+-----------------+ +-----------------
↪-+|
                                    +-----------------------------------------
↪--+
```

A tree modeled in this fashion can take advantage of the root_required query parameter to return only allocation candidates from trees which possess (or do not possess) specific traits on their root provider. For example, to return allocation candidates including VCPU with the HW_CPU_X86_AVX2 instruction set from hosts capable of COMPUTE_VOLUME_MULTI_ATTACH, a request may look like:

```
GET /allocation_candidates
  ?resources1=VCPU:1,MEMORY_MB:512&required1=HW_CPU_X86_AVX2
  &resources2=DISK_GB:100
  &group_policy=none
  &root_required=COMPUTE_VOLUME_MULTI_ATTACH
```

This will return results from both NUMA_CN and NON_NUMA_CN because both have the COMPUTE_VOLUME_MULTI_ATTACH trait on the root provider; but only NUMA2 has HW_CPU_X86_AVX2 so there will only be one result from NUMA_CN.

1. NON_NUMA_CN (VCPU, MEMORY_MB, DISK_GB)
2. NUMA_CN (DISK_GB) + NUMA2 (VCPU, MEMORY_MB)

To restrict allocation candidates to only those not in your CUSTOM_WINDOWS_LICENSE_POOL, a request may look like:

```
GET /allocation_candidates
  ?resources1=VCPU:1,MEMORY_MB:512
  &resources2=DISK_GB:100
  &group_policy=none
  &root_required=!CUSTOM_WINDOWS_LICENSE_POOL
```

This will return results only from NUMA_CN because NON_NUMA_CN has the forbidden CUSTOM_WINDOWS_LICENSE_POOL on the root provider.

1. NUMA_CN (DISK_GB) + NUMA1 (VCPU, MEMORY_MB)
2. NUMA_CN (DISK_GB) + NUMA2 (VCPU, MEMORY_MB)

The syntax of the root_required query parameter is identical to that of required[$S]: multiple trait strings may be specified, separated by commas, each optionally prefixed with ! to indicate that it is forbidden.

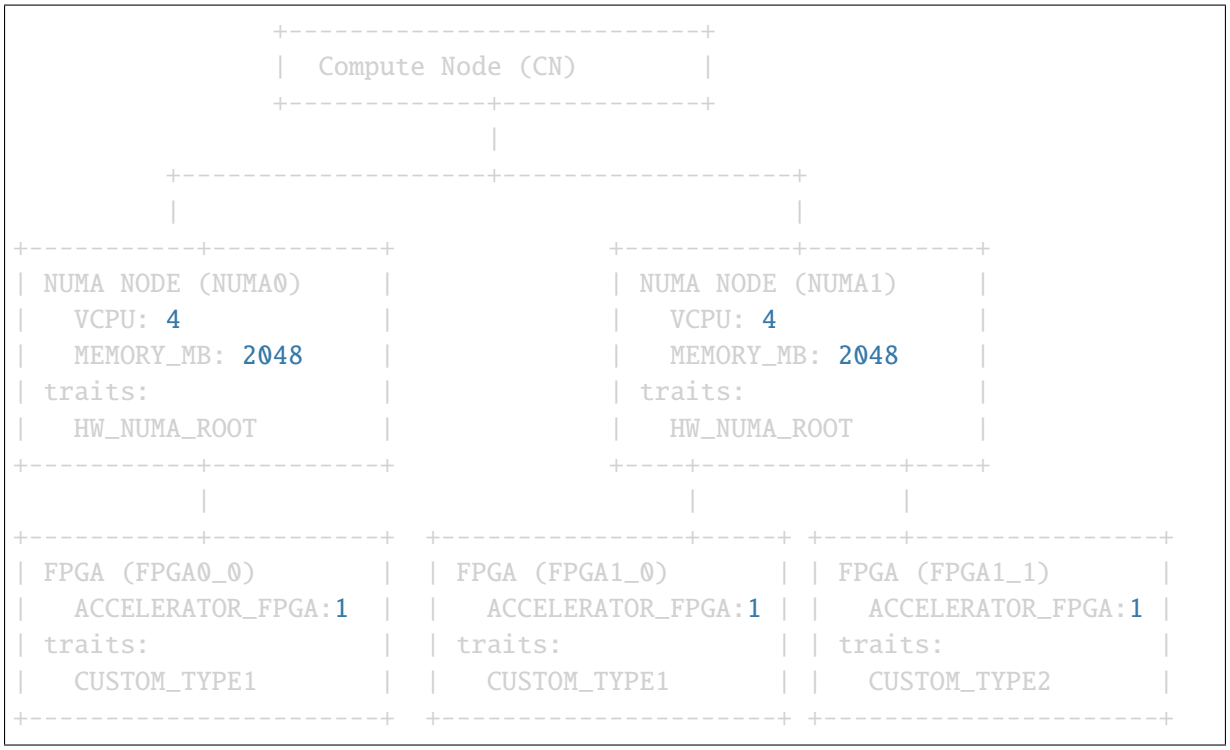
Note: root_required may not be suffixed, and may be specified only once, as it applies only to the root provider.

Note: When sharing providers are involved in the request, root_required applies only to the root of the non-sharing provider tree.

Filtering by Same Subtree

If you want to express affinity among allocations in separate request groups, use the same_subtree query parameter. It accepts a comma-separated list of request group suffix strings ($S). Each must exactly match a suffix on a granular group somewhere else in the request. If this is provided, at least one of the resource providers satisfying a specified request group must be an ancestor of the rest.

For example, given a model like:



To request FPGAs on the same NUMA node with VCPUs and MEMORY, a request may look like:

```
GET /allocation_candidates
  ?resources_COMPUTE=VCPU:1,MEMORY_MB:256
  &resources_ACCEL=ACCELERATOR_FPGA:1
  &group_policy=none
  &same_subtree=_COMPUTE,_ACCEL
```

This will produce candidates including:

1. NUMA0 (VCPU, MEMORY_MB) + FPGA0_0 (ACCELERATOR_FPGA)
2. NUMA1 (VCPU, MEMORY_MB) + FPGA1_0 (ACCELERATOR_FPGA)
3. NUMA1 (VCPU, MEMORY_MB) + FPGA1_1 (ACCELERATOR_FPGA)

but not:

4. NUMA0 (VCPU, MEMORY_MB) + FPGA1_0 (ACCELERATOR_FPGA)
5. NUMA0 (VCPU, MEMORY_MB) + FPGA1_1 (ACCELERATOR_FPGA)
6. NUMA1 (VCPU, MEMORY_MB) + FPGA0_0 (ACCELERATOR_FPGA)

The request groups specified in the same_subtree need not have a resources$S. For example, to request 2 FPGAs with different traits on the same NUMA node, a request may look like:

```
GET /allocation_candidates
  ?required_NUMA=HW_NUMA_ROOT
  &resources_ACCEL1=ACCELERATOR_FPGA:1
  &required_ACCEL1=CUSTOM_TYPE1
  &resources_ACCEL2=ACCELERATOR_FPGA:1
  &required_ACCEL2=CUSTOM_TYPE2
  &group_policy=none
  &same_subtree=_NUMA,_ACCEL1,_ACCEL2
```

This will produce candidates including:

1. `FPGA1_0` (`ACCELERATOR_FPGA`) + `FPGA1_1` (`ACCELERATOR_FPGA`) + `NUMA1`

but not:

2. `FPGA0_0` (`ACCELERATOR_FPGA`) + `FPGA1_1` (`ACCELERATOR_FPGA`) + `NUMA0`
3. `FPGA0_0` (`ACCELERATOR_FPGA`) + `FPGA1_1` (`ACCELERATOR_FPGA`) + `NUMA1`
4. `FPGA1_0` (`ACCELERATOR_FPGA`) + `FPGA1_1` (`ACCELERATOR_FPGA`) + `NUMA0`

The resource provider that satisfies the resourceless request group `?required_NUMA=HW_NUMA_ROOT`, `NUMA1` in the first example above, will not be in the `allocation_request` field of the response, but is shown in the `mappings` field.

The `same_subtree` query parameter can be repeated and each repeat group is treated independently.

1.1.2 REST API

The placement API service provides a well-documented, JSON-based HTTP API and data model. It is designed to be easy to use from whatever HTTP client is suitable. There is a plugin to the openstackclient command line tool called osc-placement which is useful for occasional inspection and manipulation of the resources in the placement service.

Microversions

The placement API uses microversions for making incremental changes to the API which client requests must opt into.

It is especially important to keep in mind that nova-compute is a client of the placement REST API and based on how Nova supports rolling upgrades the nova-compute service could be Newton level code making requests to an Ocata placement API, and vice-versa, an Ocata compute service in a cells v2 cell could be making requests to a Newton placement API.

This history of placement microversions may be found in the following subsection.

REST API Version History

This documents the changes made to the REST API with every microversion change. The description for each version should be a verbose one which has enough information to be suitable for use in user documentation.

Newton

1.0 - Initial Version

New in version Newton.

This is the initial version of the placement REST API that was released in Nova 14.0.0 (Newton). This contains the following routes:

- `/resource_providers`
- `/resource_providers/allocations`
- `/resource_providers/inventories`
- `/resource_providers/usages`
- `/allocations`

Ocata

1.1 - Resource provider aggregates

New in version Ocata.

The 1.1 version adds support for associating aggregates with resource providers.

The following new operations are added:

`GET /resource_providers/{uuid}/aggregates` Return all aggregates associated with a resource provider

`PUT /resource_providers/{uuid}/aggregates` Update the aggregates associated with a resource provider

1.2 - Add custom resource classes

New in version Ocata.

Placement API version 1.2 adds basic operations allowing an admin to create, list and delete custom resource classes.

The following new routes are added:

`GET /resource_classes` Return all resource classes

`POST /resource_classes` Create a new custom resource class

`PUT /resource_classes/{name}` Update the name of a custom resource class

DELETE /resource_classes/{name} Delete a custom resource class

GET /resource_classes/{name} Get a single resource class

Custom resource classes must begin with the prefix `CUSTOM_` and contain only the letters A through Z, the numbers 0 through 9 and the underscore _ character.

1.3 - 'member_of' query parameter

New in version Ocata.

Version 1.3 adds support for listing resource providers that are members of any of the list of aggregates provided using a `member_of` query parameter:

```
?member_of=in:{agg1_uuid},{agg2_uuid},{agg3_uuid}
```

1.4 - Filter resource providers by requested resource capacity

New in version Ocata.

The 1.4 version adds support for querying resource providers that have the ability to serve a requested set of resources. A new "resources" query string parameter is now accepted to the `GET /resource_providers` API call. This parameter indicates the requested amounts of various resources that a provider must have the capacity to serve. The "resources" query string parameter takes the form:

```
?resources=$RESOURCE_CLASS_NAME:$AMOUNT,$RESOURCE_CLASS_NAME:$AMOUNT
```

For instance, if the user wishes to see resource providers that can service a request for 2 vCPUs, 1024 MB of RAM and 50 GB of disk space, the user can issue a request to:

```
GET /resource_providers?resources=VCPU:2,MEMORY_MB:1024,DISK_GB:50
```

If the resource class does not exist, then it will return a HTTP 400.

Note: The resources filtering is also based on the *min_unit*, *max_unit* and *step_size* of the inventory record. For example, if the *max_unit* is 512 for the DISK_GB inventory for a particular resource provider and a GET request is made for *DISK_GB:1024*, that resource provider will not be returned. The *min_unit* is the minimum amount of resource that can be requested for a given inventory and resource provider. The *step_size* is the increment of resource that can be requested for a given resource on a given provider.

Pike

1.5 - 'DELETE' all inventory for a resource provider

New in version Pike.

Placement API version 1.5 adds DELETE method for deleting all inventory for a resource provider. The following new method is supported:

DELETE /resource_providers/{uuid}/inventories Delete all inventories for a given resource provider

1.6 - Traits API

New in version Pike.

The 1.6 version adds basic operations allowing an admin to create, list, and delete custom traits, also adds basic operations allowing an admin to attach traits to a resource provider.

The following new routes are added:

GET /traits Return all resource classes.

PUT /traits/{name} Insert a single custom trait.

GET /traits/{name} Check if a trait name exists.

DELETE /traits/{name} Delete the specified trait.

GET /resource_providers/{uuid}/traits Return all traits associated with a specific resource provider.

PUT /resource_providers/{uuid}/traits Update all traits for a specific resource provider.

DELETE /resource_providers/{uuid}/traits Remove any existing trait associations for a specific resource provider

Custom traits must begin with the prefix `CUSTOM_` and contain only the letters A through Z, the numbers 0 through 9 and the underscore _ character.

1.7 - Idempotent 'PUT /resource_classes/{name}'

New in version Pike.

The 1.7 version changes handling of `PUT /resource_classes/{name}` to be a create or verification of the resource class with `{name}`. If the resource class is a custom resource class and does not already exist it will be created and a `201` response code returned. If the class already exists the response code will be `204`. This makes it possible to check or create a resource class in one request.

1.8 - Require placement 'project_id', 'user_id' in 'PUT /allocations'

New in version Pike.

The 1.8 version adds `project_id` and `user_id` required request parameters to `PUT /allocations`.

1.9 - Add 'GET /usages'

New in version Pike.

The 1.9 version adds usages that can be queried by a project or project/user.

The following new routes are added:

`GET /usages?project_id=<project_id>` Return all usages for a given project.

`GET /usages?project_id=<project_id>&user_id=<user_id>` Return all usages for a given project and user.

1.10 - Allocation candidates

New in version Pike.

The 1.10 version brings a new REST resource endpoint for getting a list of allocation candidates. Allocation candidates are collections of possible allocations against resource providers that can satisfy a particular request for resources.

Queens

1.11 - Add 'allocations' link to the 'GET /resource_providers' response

New in version Queens.

The `/resource_providers/{rp_uuid}/allocations` endpoint has been available since version 1.0, but was not listed in the `links` section of the `GET /resource_providers` response. The link is included as of version 1.11.

1.12 - 'PUT' dict format to '/allocations/{consumer_uuid}'

New in version Queens.

In version 1.12 the request body of a `PUT /allocations/{consumer_uuid}` is expected to have an `object` for the `allocations` property, not as `array` as with earlier microversions. This puts the request body more in alignment with the structure of the `GET /allocations/{consumer_uuid}` response body. Because the `PUT` request requires `user_id` and `project_id` in the request body, these fields are added to the `GET` response. In addition, the response body for `GET /allocation_candidates` is updated so the allocations in the `alocation_requests` object work with the new `PUT` format.

1.13 - 'POST' multiple allocations to '/allocations'

New in version Queens.

Version 1.13 gives the ability to set or clear allocations for more than one consumer UUID with a request to `POST /allocations`.

1.14 - Add nested resource providers

New in version Queens.

The 1.14 version introduces the concept of nested resource providers. The resource provider resource now contains two new attributes:

- `parent_provider_uuid` indicates the provider's direct parent, or null if there is no parent. This attribute can be set in the call to `POST /resource_providers` and `PUT /resource_providers/{uuid}` if the attribute has not already been set to a non-NULL value (i.e. we do not support "reparenting" a provider)
- `root_provider_uuid` indicates the UUID of the root resource provider in the provider's tree. This is a read-only attribute

A new `in_tree=<UUID>` parameter is now available in the `GET /resource-providers` API call. Supplying a UUID value for the `in_tree` parameter will cause all resource providers within the "provider tree" of the provider matching `<UUID>` to be returned.

1.15 - Add 'last-modified' and 'cache-control' headers

New in version Queens.

Throughout the API, 'last-modified' headers have been added to GET responses and those PUT and POST responses that have bodies. The value is either the actual last modified time of the most recently modified associated database entity or the current time if there is no direct mapping to the database. In addition, 'cache-control: no-cache' headers are added where the 'last-modified' header has been added to prevent inadvertent caching of resources.

1.16 - Limit allocation candidates

New in version Queens.

Add support for a `limit` query parameter when making a `GET /allocation_candidates` request. The parameter accepts an integer value, `N`, which limits the maximum number of candidates returned.

1.17 - Add 'required' parameter to the allocation candidates

New in version Queens.

Add the `required` parameter to the `GET /allocation_candidates` API. It accepts a list of traits separated by `,`. The provider summary in the response will include the attached traits also.

Rocky

1.18 - Support '?required=<traits>' queryparam on 'GET /resource_providers'

New in version Rocky.

Add support for the `required` query parameter to the `GET /resource_providers` API. It accepts a comma-separated list of string trait names. When specified, the API results will be filtered to include only resource providers marked with all the specified traits. This is in addition to (logical AND) any filtering based on other query parameters.

Trait names which are empty, do not exist, or are otherwise invalid will result in a 400 error.

1.19 - Include generation and conflict detection in provider aggregates APIs

New in version Rocky.

Enhance the payloads for the `GET /resource_providers/{uuid}/aggregates` response and the `PUT /resource_providers/{uuid}/aggregates` request and response to be identical, and to include the `resource_provider_generation`. As with other generation-aware APIs, if the `resource_provider_generation` specified in the `PUT` request does not match the generation known by the server, a 409 Conflict error is returned.

1.20 - Return 200 with provider payload from 'POST /resource_providers'

New in version Rocky.

The `POST /resource_providers` API, on success, returns 200 with a payload representing the newly-created resource provider, in the same format as the corresponding `GET /resource_providers/{uuid}` call. This is to allow the caller to glean automatically-set fields, such as UUID and generation, without a subsequent GET.

1.21 - Support '?member_of=<aggregates>' queryparam on 'GET /allocation_candidates'

New in version Rocky.

Add support for the `member_of` query parameter to the `GET /allocation_candidates` API. It accepts a comma-separated list of UUIDs for aggregates. Note that if more than one aggregate UUID is passed, the comma-separated list must be prefixed with the "in:" operator. If this parameter is provided, the only resource providers returned will be those in one of the specified aggregates that meet the other parts of the request.

1.22 - Support forbidden traits on resource providers and allocations candidates

New in version Rocky.

Add support for expressing traits which are forbidden when filtering `GET /resource_providers` or `GET /allocation_candidates`. A forbidden trait is a properly formatted trait in the existing `required` parameter, prefixed by a `!`. For example `required=!STORAGE_DISK_SSD` asks that the results not include any resource providers that provide solid state disk.

1.23 - Include 'code' attribute in JSON error responses

New in version Rocky.

JSON formatted error responses gain a new attribute, `code`, with a value that identifies the type of this error. This can be used to distinguish errors that are different but use the same HTTP status code. Any error response which does not specifically define a code will have the code `placement.undefined_code`.

1.24 - Support multiple '?member_of' queryparams

New in version Rocky.

Add support for specifying multiple `member_of` query parameters to the `GET /resource_providers` API. When multiple `member_of` query parameters are found, they are AND'd together in the final query. For example, issuing a request for `GET /resource_providers?member_of=agg1&member_of=agg2` means get the resource providers that are associated with BOTH agg1 and agg2. Issuing a request for `GET /resource_providers?member_of=in:agg1,agg2&member_of=agg3` means get the resource providers that are associated with agg3 and are also associated with *any of* (agg1, agg2).

1.25 - Granular resource requests to 'GET /allocation_candidates'

New in version Rocky.

`GET /allocation_candidates` is enhanced to accept numbered groupings of resource, required/forbidden trait, and aggregate association requests. A `resources` query parameter key with a positive integer suffix (e.g. `resources42`) will be logically associated with `required` and/or `member_of` query parameter keys with the same suffix (e.g. `required42`, `member_of42`). The resources, required/forbidden traits, and aggregate associations in that group will be satisfied by the same resource provider in the response. When more than one numbered grouping is supplied, the `group_policy` query parameter is required to indicate how the groups should interact. With `group_policy=none`, separate groupings - numbered or unnumbered - may or may not be satisfied by the same provider. With `group_policy=isolate`, numbered groups are guaranteed to be satisfied by *different* providers - though there may still be overlap with the unnumbered group. In all cases, each `allocation_request` will be satisfied by providers in a single non-sharing provider tree and/or sharing providers associated via aggregate with any of the providers in that tree.

The `required` and `member_of` query parameters for a given group are optional. That is, you may specify `resources42=XXX` without a corresponding `required42=YYY` or `member_of42=ZZZ`. However, the reverse (specifying `required42=YYY` or `member_of42=ZZZ` without `resources42=XXX`) will result in an error.

The semantic of the (unnumbered) `resources`, `required`, and `member_of` query parameters is unchanged: the resources, traits, and aggregate associations specified thereby may be satisfied by any provider in the same non-sharing tree or associated via the specified aggregate(s).

1.26 - Allow inventories to have reserved value equal to total

New in version Rocky.

Starting with this version, it is allowed to set the reserved value of the resource provider inventory to be equal to total.

1.27 - Include all resource class inventories in 'provider_summaries'

New in version Rocky.

Include all resource class inventories in the `provider_summaries` field in response of the `GET /allocation_candidates` API even if the resource class is not in the requested resources.

1.28 - Consumer generation support

New in version Rocky.

A new generation field has been added to the consumer concept. Consumers are the actors that are allocated resources in the placement API. When an allocation is created, a consumer UUID is specified. Starting with microversion 1.8, a project and user ID are also required. If using microversions prior to 1.8, these are populated from the `incomplete_consumer_project_id` and `incomplete_consumer_user_id` config options from the `[placement]` section.

The consumer generation facilitates safe concurrent modification of an allocation.

A consumer generation is now returned from the following URIs:

`GET /resource_providers/{uuid}/allocations`

The response continues to be a dict with a key of `allocations`, which itself is a dict, keyed by consumer UUID, of allocations against the resource provider. For each of those dicts, a `consumer_generation` field will now be shown.

`GET /allocations/{consumer_uuid}`

The response continues to be a dict with a key of `allocations`, which itself is a dict, keyed by resource provider UUID, of allocations being consumed by the consumer with the `{consumer_uuid}`. The top-level dict will also now contain a `consumer_generation` field.

The value of the `consumer_generation` field is opaque and should only be used to send back to subsequent operations on the consumer's allocations.

The `PUT /allocations/{consumer_uuid}` URI has been modified to now require a `consumer_generation` field in the request payload. This field is required to be `null` if the caller expects that there are no allocations already existing for the consumer. Otherwise, it should contain the generation that the caller understands the consumer to be at the time of the call.

A `409 Conflict` will be returned from `PUT /allocations/{consumer_uuid}` if there was a mismatch between the supplied generation and the consumer's generation as known by the server. Similarly,

a `409 Conflict` will be returned if during the course of replacing the consumer's allocations another process concurrently changed the consumer's allocations. This allows the caller to react to the concurrent write by re-reading the consumer's allocations and re-issuing the call to replace allocations as needed.

The `PUT /allocations/{consumer_uuid}` URI has also been modified to accept an empty allocations object, thereby bringing it to parity with the behaviour of `POST /allocations`, which uses an empty allocations object to indicate that the allocations for a particular consumer should be removed. Passing an empty allocations object along with a `consumer_generation` makes `PUT /allocations/{consumer_uuid}` a **safe** way to delete allocations for a consumer. The `DELETE /allocations/{consumer_uuid}` URI remains unsafe to call in deployments where multiple callers may simultaneously be attempting to modify a consumer's allocations.

The `POST /allocations` URI variant has also been changed to require a `consumer_generation` field in the request payload **for each consumer involved in the request**. Similar responses to `PUT /allocations/{consumer_uuid}` are returned when any of the consumers generations conflict with the server's view of those consumers or if any of the consumers involved in the request are modified by another process.

Warning: In all cases, it is absolutely **NOT SAFE** to create and modify allocations for a consumer using different microversions where one of the microversions is prior to 1.28. The only way to safely modify allocations for a consumer and satisfy expectations you have regarding the prior existence (or lack of existence) of those allocations is to always use microversion 1.28+ when calling allocations API endpoints.

1.29 - Support allocation candidates with nested resource providers

New in version Rocky.

Add support for nested resource providers with the following two features. 1) `GET /allocation_candidates` is aware of nested providers. Namely, when provider trees are present, `allocation_requests` in the response of `GET /allocation_candidates` can include allocations on combinations of multiple resource providers in the same tree. 2) `root_provider_uuid` and `parent_provider_uuid` are added to `provider_summaries` in the response of `GET /allocation_candidates`.

1.30 - Provide a '/reshaper' resource

New in version Rocky.

Add support for a `POST /reshaper` resource that provides for atomically migrating resource provider inventories and associated allocations when some of the inventory moves from one resource provider to another, such as when a class of inventory moves from a parent provider to a new child provider.

Note: This is a special operation that should only be used in rare cases of resource provider topology changing when inventory is in use. Only use this if you are really sure of what you are doing.

Stein

1.31 - Add 'in_tree' queryparam on 'GET /allocation_candidates'

New in version Stein.

Add support for the `in_tree` query parameter to the `GET /allocation_candidates` API. It accepts a UUID for a resource provider. If this parameter is provided, the only resource providers returned will be those in the same tree with the given resource provider. The numbered syntax `in_tree<N>` is also supported. This restricts providers satisfying the Nth granular request group to the tree of the specified provider. This may be redundant with other `in_tree<N>` values specified in other groups (including the unnumbered group). However, it can be useful in cases where a specific resource (e.g. DISK_GB) needs to come from a specific sharing provider (e.g. shared storage).

For example, a request for `VCPU` and `VGPU` resources from `myhost` and `DISK_GB` resources from `sharing1` might look like:

```
?resources=VCPU:1&in_tree=<myhost_uuid>
&resources1=VGPU:1&in_tree1=<myhost_uuid>
&resources2=DISK_GB:100&in_tree2=<sharing1_uuid>
```

Train

1.32 - Support forbidden aggregates

New in version Train.

Add support for forbidden aggregates in `member_of` queryparam in `GET /resource_providers` and `GET /allocation_candidates`. Forbidden aggregates are prefixed with a `!`.

This negative expression can also be used in multiple `member_of` parameters:

```
?member_of=in:<agg1>,<agg2>&member_of=<agg3>&member_of=!<agg4>
```

would translate logically to

"Candidate resource providers must be at least one of agg1 or agg2, definitely in agg3 and definitely *not* in agg4."

We do NOT support `!` within the `in:` list:

```
?member_of=in:<agg1>,<agg2>,!<agg3>
```

but we support `!in:` prefix:

```
?member_of=!in:<agg1>,<agg2>,<agg3>
```

which is equivalent to:

```
?member_of=!<agg1>&member_of=!<agg2>&member_of=!<agg3>``
```

where candidate resource providers must not be in agg1, agg2, or agg3.

1.33 - Support string request group suffixes

New in version Train.

The syntax for granular groupings of resource, required/forbidden trait, and aggregate association requests introduced in `1.25` has been extended to allow, in addition to numbers, strings from 1 to 64 characters in length consisting of a-z, A-Z, 0-9, _, and -. This is done to allow naming conventions (e.g., `resources_COMPUTE` and `resources_NETWORK`) to emerge in situations where multiple services are collaborating to make requests.

For example, in addition to the already supported:

```
resources42=XXX&required42=YYY&member_of42=ZZZ
```

it is now possible to use more complex strings, including UUIDs:

```
resources_PORT_fccc7adb-095e-4bfd-8c9b-942f41990664=XXX
&required_PORT_fccc7adb-095e-4bfd-8c9b-942f41990664=YYY
&member_of_PORT_fccc7adb-095e-4bfd-8c9b-942f41990664=ZZZ
```

1.34 - Request group mappings in allocation candidates

New in version Train.

The body of the response to a `GET /allocation_candidates` request has been extended to include a `mappings` field with each allocation request. The value is a dictionary associating request group suffixes with the uuids of those resource providers that satisfy the identified request group. For convenience, this mapping can be included in the request payload for `POST /allocations`, `PUT /allocations/{consumer_uuid}`, and `POST /reshaper`, but it will be ignored.

1.35 - Support 'root_required' queryparam on GET /allocation_candidates

New in version Train.

Add support for the `root_required` query parameter to the `GET /allocation_candidates` API. It accepts a comma-delimited list of trait names, each optionally prefixed with `!` to indicate a forbidden trait, in the same format as the `required` query parameter. This restricts allocation requests in the response to only those whose (non-sharing) tree's root resource provider satisfies the specified trait requirements. See *Filtering by Root Provider Traits* for details.

1.36 - Support 'same_subtree' queryparam on GET /allocation_candidates

New in version Train.

Add support for the `same_subtree` query parameter to the `GET /allocation_candidates` API. It accepts a comma-separated list of request group suffix strings $S. Each must exactly match a suffix on a granular group somewhere else in the request. Importantly, the identified request groups need not have a resources$S. If this is provided, at least one of the resource providers satisfying a specified request group must be an ancestor of the rest. The `same_subtree` query parameter can be repeated and each repeat group is treated independently.

Xena

1.37 - Allow re-parenting and un-parenting via PUT /resource_providers/{uuid}

New in version Xena.

Add support for re-parenting and un-parenting a resource provider via `PUT /resource_providers/{uuid}` API by allowing changing the `parent_provider_uuid` to any existing provider, except providers in same subtree. Un-parenting can be achieved by setting the `parent_provider_uuid` to `null`. This means that the provider becomes a new root provider.

1.38 - Support consumer_type in allocations, usage and reshaper

New in version Xena.

Adds support for a `consumer_type` (required) key in the request body of `POST /allocations`, `PUT /allocations/{consumer_uuid}` and in the response of `GET /allocations/{consumer_uuid}`. `GET /usages` requests gain a `consumer_type` key as an optional query parameter to filter usages based on consumer_types. `GET /usages` response will group results based on the consumer type and will include a new `consumer_count` key per type irrespective of whether the `consumer_type` was specified in the request. If an `all` `consumer_type` key is provided, all results are grouped under one key, `all`. Older allocations which were not created with a consumer type are considered to have an `unknown` `consumer_type`. If an `unknown` `consumer_type` key is provided, all results are grouped under one key, `unknown`.

The corresponding changes to `POST /reshaper` are included.

CHAPTER

TWO

COMMAND LINE INTERFACE

2.1 Command-line Utilities

In this section you will find information on placement's command line utilities:

2.1.1 placement-manage

Synopsis

```
placement-manage <category> <action>
```

Description

placement-manage is used to perform administrative tasks with the placement service. It is designed for use by operators and deployers.

Options

The standard pattern for executing a `placement-manage` command is:

```
placement-manage  [-h] [--config-dir DIR] [--config-file PATH]
                  <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
placement-manage
```

You can also run with a category argument such as `db` to see a list of all commands in that category:

```
placement-manage db
```

Configuration options (for example the `[placement_database]/connection` URL) are by default found in a file at `/etc/placement/placement.conf`. The `config-dir` and `config-file` arguments may be used to select a different file.

The following sections describe the available categories and arguments for placement-manage.

Placement Database

placement-manage db version Print the current database version.

placement-manage db sync Upgrade the database schema to the most recent version. The local database connection is determined by `[placement_database]/connection` in the configuration file used by placement-manage. If the `connection` option is not set, the command will fail. The defined database must already exist.

placement-manage db stamp <version> Stamp the revision table with the given revision; dont run any migrations. This can be used when the database already exists and you want to bring it under alembic control.

placement-manage db online_data_migrations [--max-count] Perform data migration to update all live data.

`--max-count` controls the maximum number of objects to migrate in a given call. If not specified, migration will occur in batches of 50 until fully complete.

Returns exit code 0 if no (further) updates are possible, 1 if the `--max-count` option was used and some updates were completed successfully (even if others generated errors), 2 if some updates generated errors and no other migrations were able to take effect in the last batch attempted, or 127 if invalid input is provided (e.g. non-numeric max-count).

This command should be called after upgrading database schema and placement services on all controller nodes. If it exits with partial updates (exit status 1) it should be called again, even if some updates initially generated errors, because some updates may depend on others having completed. If it exits with status 2, intervention is required to resolve the issue causing remaining updates to fail. It should be considered successfully completed only when the exit status is 0.

For example:

```
$ placement-manage db online_data_migrations
Running batches of 50 until complete
2 rows matched query create_incomplete_consumers, 2 migrated
+-------------------------------------------+-------------+-----------+
|                 Migration                 | Total Found | Completed |
+-------------------------------------------+-------------+-----------+
|            set_root_provider_ids          |      0      |     0     |
|         create_incomplete_consumers       |      2      |     2     |
+-------------------------------------------+-------------+-----------+
```

In the above example, the `create_incomplete_consumers` migration found two candidate records which required a data migration. Since `--max-count` defaults to 50 and only two records were migrated with no more candidates remaining, the command completed successfully with exit code 0.

2.1.2 placement-status

Synopsis

```
placement-status <category> <command> [<args>]
```

Description

placement-status is a tool that provides routines for checking the status of a Placement deployment.

Options

The standard pattern for executing a **placement-status** command is:

```
placement-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
placement-status
```

Categories are:

- upgrade

Detailed descriptions are below.

You can also run with a category argument such as upgrade to see a list of all commands in that category:

```
placement-status upgrade
```

These sections describe the available categories and arguments for **placement-status**.

Upgrade

placement-status upgrade check Performs a release-specific readiness check before restarting services with new code. This command expects to have complete configuration and access to databases and services.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

1.0.0 (Stein)

- Checks were added for incomplete consumers and missing root provider ids both of which can be remedied by running the `placement-manage db online_data_migrations` command.

2.0.0 (Train)

- The `Missing Root Provider IDs` upgrade check will now result in a failure if there are still `resource_providers` records with a null `root_provider_id` value.

CHAPTER

THREE

CONFIGURATION

3.1 Configuration Guide

The static configuration for Placement lives in two main files: `placement.conf` and `policy.yaml`. These are described below.

3.1.1 Configuration

- *Config Reference*: A complete reference of all configuration options available in the `placement.conf` file.
- *Sample Config File*: A sample config file with inline documentation.

3.1.2 Policy

Placement, like most OpenStack projects, uses a policy language to restrict permissions on REST API actions.

- *Policy Reference*: A complete reference of all policy points in placement and what they impact.
- *Sample Policy File*: A sample placement policy file with inline documentation.

Placement Policies

> **Warning:** JSON formatted policy file is deprecated since Placement 5.0.0 (Wallaby). The oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

The following is an overview of all available policies in Placement. For a sample configuration file, refer to *Sample Placement Policy File*.

placement

`admin_api`

Default `role:admin`

Scope Types

- **system**

Default rule for most placement APIs.

`system_admin_api`

Default `role:admin and system_scope:all`

Default rule for System Admin APIs.

`system_reader_api`

Default `role:reader and system_scope:all`

Default rule for System level read only APIs.

`project_reader_api`

Default `role:reader and project_id:%(project_id)s`

Default rule for Project level read only APIs.

`system_or_project_reader`

Default `rule:system_reader_api or rule:project_reader_api`

Default rule for System+Project read only APIs.

`placement:resource_providers:list`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers`

Scope Types

- **system**

List resource providers.

`placement:resource_providers:create`

Default `rule:system_admin_api`

Operations

- **POST** `/resource_providers`

Scope Types

- **system**

Create resource provider.

`placement:resource_providers:show`

Default `rule:system_reader_api`

Operations

- **GET** /resource_providers/{uuid}

Scope Types

- **system**

Show resource provider.

`placement:resource_providers:update`

Default rule:system_admin_api

Operations

- **PUT** /resource_providers/{uuid}

Scope Types

- **system**

Update resource provider.

`placement:resource_providers:delete`

Default rule:system_admin_api

Operations

- **DELETE** /resource_providers/{uuid}

Scope Types

- **system**

Delete resource provider.

`placement:resource_classes:list`

Default rule:system_reader_api

Operations

- **GET** /resource_classes

Scope Types

- **system**

List resource classes.

`placement:resource_classes:create`

Default rule:system_admin_api

Operations

- **POST** /resource_classes

Scope Types

- **system**

Create resource class.

`placement:resource_classes:show`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_classes/{name}`

Scope Types

- **system**

Show resource class.

`placement:resource_classes:update`

Default `rule:system_admin_api`

Operations

- **PUT** `/resource_classes/{name}`

Scope Types

- **system**

Update resource class.

`placement:resource_classes:delete`

Default `rule:system_admin_api`

Operations

- **DELETE** `/resource_classes/{name}`

Scope Types

- **system**

Delete resource class.

`placement:resource_providers:inventories:list`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers/{uuid}/inventories`

Scope Types

- **system**

List resource provider inventories.

`placement:resource_providers:inventories:create`

Default `rule:system_admin_api`

Operations

- **POST** `/resource_providers/{uuid}/inventories`

Scope Types

- **system**

Create one resource provider inventory.

`placement:resource_providers:inventories:show`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers/{uuid}/inventories/{resource_class}`

Scope Types

- **system**

Show resource provider inventory.

`placement:resource_providers:inventories:update`

Default `rule:system_admin_api`

Operations

- **PUT** `/resource_providers/{uuid}/inventories`
- **PUT** `/resource_providers/{uuid}/inventories/{resource_class}`

Scope Types

- **system**

Update resource provider inventory.

`placement:resource_providers:inventories:delete`

Default `rule:system_admin_api`

Operations

- **DELETE** `/resource_providers/{uuid}/inventories`
- **DELETE** `/resource_providers/{uuid}/inventories/{resource_class}`

Scope Types

- **system**

Delete resource provider inventory.

`placement:resource_providers:aggregates:list`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers/{uuid}/aggregates`

Scope Types

- **system**

List resource provider aggregates.

`placement:resource_providers:aggregates:update`

Default `rule:system_admin_api`

Operations

- **PUT** `/resource_providers/{uuid}/aggregates`

Scope Types

- **system**

Update resource provider aggregates.

`placement:resource_providers:usages`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers/{uuid}/usages`

Scope Types

- **system**

List resource provider usages.

`placement:usages`

Default `rule:system_or_project_reader`

Operations

- **GET** `/usages`

Scope Types

- **system**
- **project**

List total resource usages for a given project.

`placement:traits:list`

Default `rule:system_reader_api`

Operations

- **GET** `/traits`

Scope Types

- **system**

List traits.

`placement:traits:show`

Default `rule:system_reader_api`

Operations

- **GET** `/traits/{name}`

Scope Types

- **system**

Show trait.

`placement:traits:update`

Default `rule:system_admin_api`

Operations

- **PUT** `/traits/{name}`

Scope Types

- **system**

Update trait.

`placement:traits:delete`

Default `rule:system_admin_api`

Operations

- **DELETE** `/traits/{name}`

Scope Types

- **system**

Delete trait.

`placement:resource_providers:traits:list`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers/{uuid}/traits`

Scope Types

- **system**

List resource provider traits.

`placement:resource_providers:traits:update`

Default `rule:system_admin_api`

Operations

- **PUT** `/resource_providers/{uuid}/traits`

Scope Types

- **system**

Update resource provider traits.

`placement:resource_providers:traits:delete`

Default `rule:system_admin_api`

Operations

- **DELETE** `/resource_providers/{uuid}/traits`

Scope Types

- **system**

Delete resource provider traits.

`placement:allocations:manage`

Default `rule:system_admin_api`

Operations

- **POST** `/allocations`

Scope Types

- **system**

Manage allocations.

`placement:allocations:list`

Default `rule:system_reader_api`

Operations

- **GET** `/allocations/{consumer_uuid}`

Scope Types

- **system**

List allocations.

`placement:allocations:update`

Default `rule:system_admin_api`

Operations

- **PUT** `/allocations/{consumer_uuid}`

Scope Types

- **system**

Update allocations.

`placement:allocations:delete`

Default `rule:system_admin_api`

Operations

- **DELETE** `/allocations/{consumer_uuid}`

Scope Types

- **system**

Delete allocations.

`placement:resource_providers:allocations:list`

Default `rule:system_reader_api`

Operations

- **GET** `/resource_providers/{uuid}/allocations`

Scope Types

- **system**

List resource provider allocations.

`placement:allocation_candidates:list`

Default `rule:system_reader_api`

Operations

- **GET** `/allocation_candidates`

Scope Types

- **system**

List allocation candidates.

`placement:reshaper:reshape`

Default `rule:system_admin_api`

Operations

- **POST** `/reshaper`

Scope Types

- **system**

Reshape Inventory and Allocations.

Sample Placement Policy File

Warning: JSON formatted policy file is deprecated since Placement 5.0.0 (Wallaby). The oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

The following is a sample placement policy file for adaptation and use.

The sample policy can also be viewed in `file form`.

Important: The sample policy file is auto-generated from placement when this documentation is built. You must ensure your version of placement matches the version of this documentation.

```
# DEPRECATED
# "admin_api" has been deprecated since W.
# Placement API policies are introducing new default roles with
# scope_type capabilities. Old policies are deprecated and silently
# going to be ignored in the placement 6.0.0 (Xena) release.
# Default rule for most placement APIs.
# Intended scope(s): system
#"admin_api": "role:admin"

# Default rule for System Admin APIs.
```

(continues on next page)

(continued from previous page)

```
#"system_admin_api": "role:admin and system_scope:all"

# DEPRECATED
# "rule:admin_api":"role:admin" has been deprecated since W in favor
# of "system_admin_api":"role:admin and system_scope:all".
# Placement API policies are introducing new default roles with
# scope_type capabilities. Old policies are deprecated and silently
# going to be ignored in the placement 6.0.0 (Xena) release.
"rule:admin_api": "rule:system_admin_api"

# Default rule for System level read only APIs.
#"system_reader_api": "role:reader and system_scope:all"

# DEPRECATED
# "rule:admin_api":"role:admin" has been deprecated since W in favor
# of "system_reader_api":"role:reader and system_scope:all".
# Placement API policies are introducing new default roles with
# scope_type capabilities. Old policies are deprecated and silently
# going to be ignored in the placement 6.0.0 (Xena) release.
"rule:admin_api": "rule:system_reader_api"

# Default rule for Project level read only APIs.
#"project_reader_api": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "rule:admin_api":"role:admin" has been deprecated since W in favor
# of "project_reader_api":"role:reader and project_id:%(project_id)s".
# Placement API policies are introducing new default roles with
# scope_type capabilities. Old policies are deprecated and silently
# going to be ignored in the placement 6.0.0 (Xena) release.
"rule:admin_api": "rule:project_reader_api"

# Default rule for System+Project read only APIs.
#"system_or_project_reader": "rule:system_reader_api or rule:project_reader_
↪api"

# DEPRECATED
# "rule:admin_api":"role:admin" has been deprecated since W in favor
# of "system_or_project_reader":"rule:system_reader_api or
# rule:project_reader_api".
# Placement API policies are introducing new default roles with
# scope_type capabilities. Old policies are deprecated and silently
# going to be ignored in the placement 6.0.0 (Xena) release.
"rule:admin_api": "rule:system_or_project_reader"

# List resource providers.
# GET  /resource_providers
# Intended scope(s): system
#"placement:resource_providers:list": "rule:system_reader_api"
```

(continues on next page)

(continued from previous page)

```
# Create resource provider.
# POST  /resource_providers
# Intended scope(s): system
#"placement:resource_providers:create": "rule:system_admin_api"

# Show resource provider.
# GET  /resource_providers/{uuid}
# Intended scope(s): system
#"placement:resource_providers:show": "rule:system_reader_api"

# Update resource provider.
# PUT  /resource_providers/{uuid}
# Intended scope(s): system
#"placement:resource_providers:update": "rule:system_admin_api"

# Delete resource provider.
# DELETE  /resource_providers/{uuid}
# Intended scope(s): system
#"placement:resource_providers:delete": "rule:system_admin_api"

# List resource classes.
# GET  /resource_classes
# Intended scope(s): system
#"placement:resource_classes:list": "rule:system_reader_api"

# Create resource class.
# POST  /resource_classes
# Intended scope(s): system
#"placement:resource_classes:create": "rule:system_admin_api"

# Show resource class.
# GET  /resource_classes/{name}
# Intended scope(s): system
#"placement:resource_classes:show": "rule:system_reader_api"

# Update resource class.
# PUT  /resource_classes/{name}
# Intended scope(s): system
#"placement:resource_classes:update": "rule:system_admin_api"

# Delete resource class.
# DELETE  /resource_classes/{name}
# Intended scope(s): system
#"placement:resource_classes:delete": "rule:system_admin_api"

# List resource provider inventories.
# GET  /resource_providers/{uuid}/inventories
# Intended scope(s): system
```

(continues on next page)

(continued from previous page)

```
#"placement:resource_providers:inventories:list": "rule:system_reader_api"

# Create one resource provider inventory.
# POST  /resource_providers/{uuid}/inventories
# Intended scope(s): system
#"placement:resource_providers:inventories:create": "rule:system_admin_api"

# Show resource provider inventory.
# GET  /resource_providers/{uuid}/inventories/{resource_class}
# Intended scope(s): system
#"placement:resource_providers:inventories:show": "rule:system_reader_api"

# Update resource provider inventory.
# PUT  /resource_providers/{uuid}/inventories
# PUT  /resource_providers/{uuid}/inventories/{resource_class}
# Intended scope(s): system
#"placement:resource_providers:inventories:update": "rule:system_admin_api"

# Delete resource provider inventory.
# DELETE  /resource_providers/{uuid}/inventories
# DELETE  /resource_providers/{uuid}/inventories/{resource_class}
# Intended scope(s): system
#"placement:resource_providers:inventories:delete": "rule:system_admin_api"

# List resource provider aggregates.
# GET  /resource_providers/{uuid}/aggregates
# Intended scope(s): system
#"placement:resource_providers:aggregates:list": "rule:system_reader_api"

# Update resource provider aggregates.
# PUT  /resource_providers/{uuid}/aggregates
# Intended scope(s): system
#"placement:resource_providers:aggregates:update": "rule:system_admin_api"

# List resource provider usages.
# GET  /resource_providers/{uuid}/usages
# Intended scope(s): system
#"placement:resource_providers:usages": "rule:system_reader_api"

# List total resource usages for a given project.
# GET  /usages
# Intended scope(s): system, project
#"placement:usages": "rule:system_or_project_reader"

# List traits.
# GET  /traits
# Intended scope(s): system
#"placement:traits:list": "rule:system_reader_api"
```

(continues on next page)

(continued from previous page)

```
# Show trait.
# GET  /traits/{name}
# Intended scope(s): system
#"placement:traits:show": "rule:system_reader_api"

# Update trait.
# PUT  /traits/{name}
# Intended scope(s): system
#"placement:traits:update": "rule:system_admin_api"

# Delete trait.
# DELETE  /traits/{name}
# Intended scope(s): system
#"placement:traits:delete": "rule:system_admin_api"

# List resource provider traits.
# GET  /resource_providers/{uuid}/traits
# Intended scope(s): system
#"placement:resource_providers:traits:list": "rule:system_reader_api"

# Update resource provider traits.
# PUT  /resource_providers/{uuid}/traits
# Intended scope(s): system
#"placement:resource_providers:traits:update": "rule:system_admin_api"

# Delete resource provider traits.
# DELETE  /resource_providers/{uuid}/traits
# Intended scope(s): system
#"placement:resource_providers:traits:delete": "rule:system_admin_api"

# Manage allocations.
# POST  /allocations
# Intended scope(s): system
#"placement:allocations:manage": "rule:system_admin_api"

# List allocations.
# GET  /allocations/{consumer_uuid}
# Intended scope(s): system
#"placement:allocations:list": "rule:system_reader_api"

# Update allocations.
# PUT  /allocations/{consumer_uuid}
# Intended scope(s): system
#"placement:allocations:update": "rule:system_admin_api"

# Delete allocations.
# DELETE  /allocations/{consumer_uuid}
# Intended scope(s): system
#"placement:allocations:delete": "rule:system_admin_api"
```

(continues on next page)

(continued from previous page)

```
# List resource provider allocations.
# GET  /resource_providers/{uuid}/allocations
# Intended scope(s): system
#"placement:resource_providers:allocations:list": "rule:system_reader_api"

# List allocation candidates.
# GET  /allocation_candidates
# Intended scope(s): system
#"placement:allocation_candidates:list": "rule:system_reader_api"

# Reshape Inventory and Allocations.
# POST  /reshaper
# Intended scope(s): system
#"placement:reshaper:reshape": "rule:system_admin_api"
```

Configuration Options

The following is an overview of all available configuration options in Placement. For a sample configuration file, refer to *Sample Configuration File*.

DEFAULT

tempdir

Type string

Default <None>

Explicitly specify the temporary working directory.

pybasedir

Type string

Default <Path>

This option has a sample default set, which means that its actual default value may vary from the one documented above.

The directory where the Placement python modules are installed.

This is the default path for other config options which need to persist Placement internal data. It is very unlikely that you need to change this option from its default value.

Possible values:

- The full path to a directory.

Related options:

- state_path

state_path

Type string

Default `$pybasedir`

The top-level directory for maintaining state used in Placement.

This directory is used to store Placement's internal state. It is used by some tests that have behaviors carried over from Nova.

Possible values:

- The full path to a directory. Defaults to value provided in `pybasedir`.

`debug`

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

`log_config_append`

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 1: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

`log_date_format`

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

`log_file`

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 2: Deprecated Variations

Group	Name
DEFAULT	logfile

`log_dir`

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 3: Deprecated Variations

Group	Name
DEFAULT	logdir

`watch_log_file`

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

`use_syslog`

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

`use_journal`

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

`syslog_log_facility`

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

`use_json`

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

`use_stderr`

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

`use_eventlog`

Type boolean

Default `False`

Log output to Windows Event Log.

`log_rotate_interval`

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is set to "interval".

`log_rotate_interval_type`

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

`max_logfile_count`

Type integer

Default `30`

Maximum number of rotated log files.

`max_logfile_size_mb`

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if "log_rotation_type" is not set to "size".

`log_rotation_type`

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

`logging_context_format_string`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

`logging_default_format_string`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

`logging_debug_format_suffix`

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

`logging_exception_prefix`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

`logging_user_identity_format`

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

`default_log_levels`

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO',`

```
'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.
urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN',
'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN',
'urllib3.util.retry=WARN', 'keystonemiddleware=WARN',
'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN',
'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO',
'dogpile.core.dogpile=INFO']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type boolean

Default `False`

Enables or disables publication of error events.

instance_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

instance_uuid_format

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

rate_limit_interval

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

rate_limit_burst

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

rate_limit_except_level

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

api

Options under this group are used to define Placement API.

`auth_strategy`

Type string

Default `keystone`

Valid Values keystone, noauth2

This determines the strategy to use for authentication: keystone or noauth2. 'noauth2' is designed for testing only, as it does no actual credential checking. 'noauth2' provides administrative credentials only if 'admin' is specified as the username.

Table 4: Deprecated Variations

Group	Name
DEFAULT	auth_strategy

cors

`allowed_origin`

Type list

Default `<None>`

Indicate whether this resource may be shared with the domain received in the requests "origin" header. Format: "<protocol>://<host>[:<port>]", no trailing slash. Example: https://horizon.example.com

`allow_credentials`

Type boolean

Default `True`

Indicate that the actual request can include user credentials

`expose_headers`

Type list

Default `[]`

Indicate which headers are safe to expose to the API. Defaults to HTTP Simple Headers.

`max_age`

Type integer

Default `3600`

Maximum cache age of CORS preflight requests.

allow_methods

Type list

Default ['OPTIONS', 'GET', 'HEAD', 'POST', 'PUT', 'DELETE', 'TRACE', 'PATCH']

Indicate which methods can be used during the actual request.

allow_headers

Type list

Default []

Indicate which header field names may be used during the actual request.

keystone_authtoken

www_authenticate_uri

Type string

Default <None>

Complete "public" Identity API endpoint. This endpoint should not be an "admin" endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If you're using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint.

Table 5: Deprecated Variations

Group	Name
keystone_authtoken	auth_uri

auth_uri

Type string

Default <None>

Complete "public" Identity API endpoint. This endpoint should not be an "admin" endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If you're using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint. This option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

Warning: This option is deprecated for removal since Queens. Its value may be silently ignored in the future.

Reason The auth_uri option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

`auth_version`

Type string

Default <None>

API version of the Identity API endpoint.

`interface`

Type string

Default `internal`

Interface to use for the Identity API endpoint. Valid values are "public", "internal" (default) or "admin".

`delay_auth_decision`

Type boolean

Default `False`

Do not handle authorization requests within the middleware, but delegate the authorization decision to downstream WSGI components.

`http_connect_timeout`

Type integer

Default <None>

Request timeout value for communicating with Identity API server.

`http_request_max_retries`

Type integer

Default `3`

How many times are we trying to reconnect when communicating with Identity API Server.

`cache`

Type string

Default <None>

Request environment key where the Swift cache object is stored. When auth_token middleware is deployed with a Swift cache, use this option to have the middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.

`certfile`

Type string

Default <None>

Required if identity server requires client certificate

`keyfile`

Type string

Default <None>

Required if identity server requires client certificate

cafile

Type string

Default <None>

A PEM encoded Certificate Authority to use when verifying HTTPs connections. Defaults to system CAs.

insecure

Type boolean

Default False

Verify HTTPS connections.

region_name

Type string

Default <None>

The region in which the identity server can be found.

memcached_servers

Type list

Default <None>

Optionally specify a list of memcached server(s) to use for caching. If left undefined, tokens will instead be cached in-process.

Table 6: Deprecated Variations

Group	Name
keystone_authtoken	memcache_servers

token_cache_time

Type integer

Default 300

In order to prevent excessive effort spent validating tokens, the middleware caches previously-seen tokens for a configurable duration (in seconds). Set to -1 to disable caching completely.

memcache_security_strategy

Type string

Default None

Valid Values None, MAC, ENCRYPT

(Optional) If defined, indicate whether token data should be authenticated or authenticated and encrypted. If MAC, token data is authenticated (with HMAC) in the cache. If ENCRYPT, token data is encrypted and authenticated in the cache. If the value is not one of these options or empty, auth_token will raise an exception on initialization.

memcache_secret_key

Type string

Default <None>

(Optional, mandatory if memcache_security_strategy is defined) This string is used for key derivation.

memcache_pool_dead_retry

Type integer

Default 300

(Optional) Number of seconds memcached server is considered dead before it is tried again.

memcache_pool_maxsize

Type integer

Default 10

(Optional) Maximum total number of open connections to every memcached server.

memcache_pool_socket_timeout

Type integer

Default 3

(Optional) Socket timeout in seconds for communicating with a memcached server.

memcache_pool_unused_timeout

Type integer

Default 60

(Optional) Number of seconds a connection to memcached is held unused in the pool before it is closed.

memcache_pool_conn_get_timeout

Type integer

Default 10

(Optional) Number of seconds that an operation will wait to get a memcached client connection from the pool.

memcache_use_advanced_pool

Type boolean

Default True

(Optional) Use the advanced (eventlet safe) memcached client pool.

include_service_catalog

Type boolean

Default True

(Optional) Indicate whether to set the X-Service-Catalog header. If False, middleware will not ask for service catalog on token validation and will not set the X-Service-Catalog header.

enforce_token_bind

Type string

Default `permissive`

Used to control the use and type of token binding. Can be set to: "disabled" to not check token binding. "permissive" (default) to validate binding information if the bind type is of a form known to the server and ignore it if not. "strict" like "permissive" but if the bind type is unknown the token will be rejected. "required" any form of token binding is needed to be allowed. Finally the name of a binding method that must be present in tokens.

`service_token_roles`

Type list

Default `['service']`

A choice of roles that must be present in a service token. Service tokens are allowed to request that an expired token can be used and so this check should tightly control that only actual services should be sending this token. Roles here are applied as an ANY check so any role in this list must be present. For backwards compatibility reasons this currently only affects the allow_expired check.

`service_token_roles_required`

Type boolean

Default `False`

For backwards compatibility reasons we must let valid service tokens pass that don't pass the service_token_roles check as valid. Setting this true will become the default in a future release and should be enabled if possible.

`service_type`

Type string

Default `<None>`

The name or type of the service as it appears in the service catalog. This is used to validate tokens that have restricted access rules.

`auth_type`

Type unknown type

Default `<None>`

Authentication type to load

Table 7: Deprecated Variations

Group	Name
keystone_authtoken	auth_plugin

`auth_section`

Type unknown type

Default `<None>`

Config Section from which to load plugin specific options

oslo_policy

enforce_scope

Type boolean

Default `False`

This option controls whether or not to enforce scope when evaluating policies. If `True`, the scope of the token used in the request is compared to the `scope_types` of the policy being enforced. If the scopes do not match, an `InvalidScope` exception will be raised. If `False`, a message will be logged informing operators that policies are being invoked with mismatching scope.

enforce_new_defaults

Type boolean

Default `False`

This option controls whether or not to use old deprecated defaults when evaluating policies. If `True`, the old deprecated defaults are not going to be evaluated. This means if any existing token is allowed for old defaults but is disallowed for new defaults, it will be disallowed. It is encouraged to enable this flag along with the `enforce_scope` flag so that you can get the benefits of new defaults and `scope_type` together

policy_file

Type string

Default `policy.json`

The relative or absolute path of a file that maps roles to permissions for a given service. Relative paths must be specified in relation to the configuration file setting this option.

Table 8: Deprecated Variations

Group	Name
DEFAULT	policy_file

policy_default_rule

Type string

Default `default`

Default rule. Enforced when a requested rule is not found.

Table 9: Deprecated Variations

Group	Name
DEFAULT	policy_default_rule

policy_dirs

Type multi-valued

Default `policy.d`

Directories where policy configuration files are stored. They can be relative to any directory in the search path defined by the config_dir option, or absolute paths. The file defined by policy_file must exist for these directories to be searched. Missing or empty directories are ignored.

Table 10: Deprecated Variations

Group	Name
DEFAULT	policy_dirs

remote_content_type

Type string

Default `application/x-www-form-urlencoded`

Valid Values application/x-www-form-urlencoded, application/json

Content Type to send and receive data for REST based policy check

remote_ssl_verify_server_crt

Type boolean

Default `False`

server identity verification for REST based policy check

remote_ssl_ca_crt_file

Type string

Default `<None>`

Absolute path to ca cert file for REST based policy check

remote_ssl_client_crt_file

Type string

Default `<None>`

Absolute path to client cert for REST based policy check

remote_ssl_client_key_file

Type string

Default `<None>`

Absolute path client key file REST based policy check

placement

randomize_allocation_candidates

Type boolean

Default `False`

If True, when limiting allocation candidate results, the results will be a random sampling of the full result set. If False, allocation candidates are returned in a deterministic but undefined order.

That is, all things being equal, two requests for allocation candidates will return the same results in the same order; but no guarantees are made as to how that order is determined.

`incomplete_consumer_project_id`

Type string

Default `00000000-0000-0000-0000-000000000000`

Early API microversions (<1.8) allowed creating allocations and not specifying a project or user identifier for the consumer. In cleaning up the data modeling, we no longer allow missing project and user information. If an older client makes an allocation, we'll use this in place of the information it doesn't provide.

`incomplete_consumer_user_id`

Type string

Default `00000000-0000-0000-0000-000000000000`

Early API microversions (<1.8) allowed creating allocations and not specifying a project or user identifier for the consumer. In cleaning up the data modeling, we no longer allow missing project and user information. If an older client makes an allocation, we'll use this in place of the information it doesn't provide.

`allocation_conflict_retry_count`

Type integer

Default `10`

The number of times to retry, server-side, writing allocations when there is a resource provider generation conflict. Raising this value may be useful when many concurrent allocations to the same resource provider are expected.

placement_database

The *Placement API Database* is a the database used with the placement service. If the connection option is not set, the placement service will not start.

`connection`

Type string

Default `<None>`

The SQLAlchemy connection string to use to connect to the database.

`connection_parameters`

Type string

Default `''`

Optional URL parameters to append onto the connection URL at connect time; specify as param1=value1&param2=value2&...

`sqlite_synchronous`

Type boolean

Default `True`

If True, SQLite uses synchronous mode.

slave_connection

Type string

Default <None>

The SQLAlchemy connection string to use to connect to the slave database.

mysql_sql_mode

Type string

Default TRADITIONAL

The SQL mode to be used for MySQL sessions. This option, including the default, overrides any server-set SQL mode. To use whatever SQL mode is set by the server configuration, set this to no value. Example: mysql_sql_mode=

connection_recycle_time

Type integer

Default 3600

Connections which have been present in the connection pool longer than this number of seconds will be replaced with a new one the next time they are checked out from the pool.

max_pool_size

Type integer

Default <None>

Maximum number of SQL connections to keep open in a pool. Setting a value of 0 indicates no limit.

max_retries

Type integer

Default 10

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

retry_interval

Type integer

Default 10

Interval between retries of opening a SQL connection.

max_overflow

Type integer

Default <None>

If set, use this value for max_overflow with SQLAlchemy.

connection_debug

Type integer

Default `0`

Verbosity of SQL debugging information: 0=None, 100=Everything.

`connection_trace`

Type boolean

Default `False`

Add Python stack traces to SQL as comment strings.

`pool_timeout`

Type integer

Default `<None>`

If set, use this value for pool_timeout with SQLAlchemy.

`sync_on_startup`

Type boolean

Default `False`

If True, database schema migrations will be attempted when the web service starts.

profiler

`enabled`

Type boolean

Default `False`

Enable the profiling for all services on this node.

Default value is False (fully disable the profiling feature).

Possible values:

- True: Enables the feature
- False: Disables the feature. The profiling cannot be started via this project operations. If the profiling is triggered by another project, this project part will be empty.

Table 11: Deprecated Variations

Group	Name
profiler	profiler_enabled

`trace_sqlalchemy`

Type boolean

Default `False`

Enable SQL requests profiling in services.

Default value is False (SQL requests won't be traced).

Possible values:

- True: Enables SQL requests profiling. Each SQL query will be part of the trace and can the be analyzed by how much time was spent for that.
- False: Disables SQL requests profiling. The spent time is only shown on a higher level of operations. Single SQL queries cannot be analyzed this way.

hmac_keys

Type string

Default `SECRET_KEY`

Secret key(s) to use for encrypting context data for performance profiling.

This string value should have the following format: <key1>[,<key2>,...<keyn>], where each key is some random string. A user who triggers the profiling via the REST API has to set one of these keys in the headers of the REST API call to include profiling results of this node for this particular project.

Both "enabled" flag and "hmac_keys" config options should be set to enable profiling. Also, to generate correct profiling information across all services at least one key needs to be consistent between OpenStack projects. This ensures it can be used from client side to generate the trace, containing information from all possible resources.

connection_string

Type string

Default `messaging://`

Connection string for a notifier backend.

Default value is `messaging://` which sets the notifier to oslo_messaging.

Examples of possible values:

- `messaging://` - use oslo_messaging driver for sending spans.
- `redis://127.0.0.1:6379` - use redis driver for sending spans.
- `mongodb://127.0.0.1:27017` - use mongodb driver for sending spans.
- `elasticsearch://127.0.0.1:9200` - use elasticsearch driver for sending spans.
- `jaeger://127.0.0.1:6831` - use jaeger tracing as driver for sending spans.

es_doc_type

Type string

Default `notification`

Document type for notification indexing in elasticsearch.

es_scroll_time

Type string

Default `2m`

This parameter is a time value parameter (for example: es_scroll_time=2m), indicating for how long the nodes that participate in the search will maintain relevant resources in order to continue and support it.

es_scroll_size

Type integer

Default `10000`

Elasticsearch splits large requests in batches. This parameter defines maximum size of each batch (for example: es_scroll_size=10000).

`socket_timeout`

Type floating point

Default `0.1`

Redissentinel provides a timeout option on the connections. This parameter defines that timeout (for example: socket_timeout=0.1).

`sentinel_service_name`

Type string

Default `mymaster`

Redissentinel uses a service name to identify a master redis service. This parameter defines the name (for example: `sentinal_service_name=mymaster`).

`filter_error_trace`

Type boolean

Default `False`

Enable filter traces that contain error/exception to a separated place.

Default value is set to False.

Possible values:

- True: Enable filter traces that contain error/exception.
- False: Disable the filter.

Sample Configuration File

The following is a sample Placement configuration for adaptation and use. For a detailed overview of all available configuration options, refer to *Configuration Options*.

The sample configuration can also be viewed in `file form`.

Important: The sample configuration file is auto-generated from placement when this documentation is built. You must ensure your version of placement matches the version of this documentation.

```
[DEFAULT]

#
# From oslo.log
#

# If set to true, the logging level will be set to DEBUG instead of the␣
↪default
```

(continues on next page)

(continued from previous page)

```
# INFO level. (boolean value)
# Note: This option can be changed without restarting.
#debug = false

# The name of a logging configuration file. This file is appended to any
# existing logging configuration files. For details about logging␣
↪configuration
# files, see the Python logging module documentation. Note that when logging
# configuration files are used then all logging configuration is set in the
# configuration file and other logging configuration options are ignored (for
# example, log-date-format). (string value)
# Note: This option can be changed without restarting.
# Deprecated group/name - [DEFAULT]/log_config
#log_config_append = <None>

# Defines the format string for %%(asctime)s in log records. Default:
# %(default)s . This option is ignored if log_config_append is set. (string
# value)
#log_date_format = %Y-%m-%d %H:%M:%S

# (Optional) Name of log file to send logging output to. If no default is set,
# logging will go to stderr as defined by use_stderr. This option is ignored␣
↪if
# log_config_append is set. (string value)
# Deprecated group/name - [DEFAULT]/logfile
#log_file = <None>

# (Optional) The base directory used for relative log_file  paths. This option
# is ignored if log_config_append is set. (string value)
# Deprecated group/name - [DEFAULT]/logdir
#log_dir = <None>

# Uses logging handler designed to watch file system. When log file is moved␣
↪or
# removed this handler will open a new log file with specified path
# instantaneously. It makes sense only if log_file option is specified and␣
↪Linux
# platform is used. This option is ignored if log_config_append is set.␣
↪(boolean
# value)
#watch_log_file = false

# Use syslog for logging. Existing syslog format is DEPRECATED and will be
# changed later to honor RFC5424. This option is ignored if log_config_append␣
↪is
# set. (boolean value)
#use_syslog = false

# Enable journald for logging. If running in a systemd environment you may␣
↪wish
```

(continues on next page)

(continued from previous page)

```
# to enable journal support. Doing so will use the journal native protocol␣
↪which
# includes structured metadata in addition to log messages.This option is
# ignored if log_config_append is set. (boolean value)
#use_journal = false

# Syslog facility to receive log lines. This option is ignored if
# log_config_append is set. (string value)
#syslog_log_facility = LOG_USER

# Use JSON formatting for logging. This option is ignored if log_config_append
# is set. (boolean value)
#use_json = false

# Log output to standard error. This option is ignored if log_config_append is
# set. (boolean value)
#use_stderr = false

# Log output to Windows Event Log. (boolean value)
#use_eventlog = false

# The amount of time before the log files are rotated. This option is ignored
# unless log_rotation_type is set to "interval". (integer value)
#log_rotate_interval = 1

# Rotation interval type. The time of the last file change (or the time when␣
↪the
# service was started) is used when scheduling the next rotation. (string␣
↪value)
# Possible values:
# Seconds - <No description provided>
# Minutes - <No description provided>
# Hours - <No description provided>
# Days - <No description provided>
# Weekday - <No description provided>
# Midnight - <No description provided>
#log_rotate_interval_type = days

# Maximum number of rotated log files. (integer value)
#max_logfile_count = 30

# Log file maximum size in MB. This option is ignored if "log_rotation_type"␣
↪is
# not set to "size". (integer value)
#max_logfile_size_mb = 200

# Log rotation type. (string value)
# Possible values:
# interval - Rotate logs at predefined time intervals.
```

(continues on next page)

(continued from previous page)

```
# size - Rotate logs once they reach a predefined size.
# none - Do not rotate log files.
#log_rotation_type = none

# Format string to use for log messages with context. Used by
# oslo_log.formatters.ContextFormatter (string value)
#logging_context_format_string = %(asctime)s.%(msecs)03d %(process)d
↪%(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s
↪%(message)s

# Format string to use for log messages when context is undefined. Used by
# oslo_log.formatters.ContextFormatter (string value)
#logging_default_format_string = %(asctime)s.%(msecs)03d %(process)d
↪%(levelname)s %(name)s [-] %(instance)s%(message)s

# Additional data to append to log message when logging level for the message␣
↪is
# DEBUG. Used by oslo_log.formatters.ContextFormatter (string value)
#logging_debug_format_suffix = %(funcName)s %(pathname)s:%(lineno)d

# Prefix each line of exception output with this format. Used by
# oslo_log.formatters.ContextFormatter (string value)
#logging_exception_prefix = %(asctime)s.%(msecs)03d %(process)d ERROR
↪%(name)s %(instance)s

# Defines the format string for %(user_identity)s that is used in
# logging_context_format_string. Used by oslo_log.formatters.ContextFormatter
# (string value)
#logging_user_identity_format = %(user)s %(tenant)s %(domain)s %(user_
↪domain)s %(project_domain)s

# List of package logging levels in logger=LEVEL pairs. This option is ignored
# if log_config_append is set. (list value)
#default_log_levels = amqp=WARN,amqplib=WARN,boto=WARN,qpid=WARN,
↪sqlalchemy=WARN,suds=INFO,oslo.messaging=INFO,oslo_messaging=INFO,
↪iso8601=WARN,requests.packages.urllib3.connectionpool=WARN,urllib3.
↪connectionpool=WARN,websocket=WARN,requests.packages.urllib3.util.
↪retry=WARN,urllib3.util.retry=WARN,keystonemiddleware=WARN,routes.
↪middleware=WARN,stevedore=WARN,taskflow=WARN,keystoneauth=WARN,oslo.
↪cache=INFO,oslo_policy=INFO,dogpile.core.dogpile=INFO

# Enables or disables publication of error events. (boolean value)
#publish_errors = false

# The format for an instance that is passed with the log message. (string␣
↪value)
#instance_format = "[instance: %(uuid)s] "

# The format for an instance UUID that is passed with the log message. (string
```

(continues on next page)

(continued from previous page)

```
# value)
#instance_uuid_format = "[instance: %(uuid)s] "

# Interval, number of seconds, of log rate limiting. (integer value)
#rate_limit_interval = 0

# Maximum number of logged messages per rate_limit_interval. (integer value)
#rate_limit_burst = 0

# Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG␣
↪or
# empty string. Logs with level greater or equal to rate_limit_except_level␣
↪are
# not filtered. An empty string means that all levels are filtered. (string
# value)
#rate_limit_except_level = CRITICAL

# Enables or disables fatal status of deprecations. (boolean value)
#fatal_deprecations = false

#
# From placement.conf
#

# Explicitly specify the temporary working directory. (string value)
#tempdir = <None>

#
# The directory where the Placement python modules are installed.
#
# This is the default path for other config options which need to persist
# Placement internal data. It is very unlikely that you need to
# change this option from its default value.
#
# Possible values:
#
# * The full path to a directory.
#
# Related options:
#
# * ``state_path``
#  (string value)
#
# This option has a sample default set, which means that
# its actual default value may vary from the one documented
# below.
#pybasedir = <Path>

#
```

(continues on next page)

(continued from previous page)

```
# The top-level directory for maintaining state used in Placement.
#
# This directory is used to store Placement's internal state. It is used by␣
↪some
# tests that have behaviors carried over from Nova.
#
# Possible values:
#
# * The full path to a directory. Defaults to value provided in ``pybasedir``.
#  (string value)
#state_path = $pybasedir


[api]
#
# Options under this group are used to define Placement API.

#
# From placement.conf
#

#
# This determines the strategy to use for authentication: keystone or noauth2.
# 'noauth2' is designed for testing only, as it does no actual credential
# checking. 'noauth2' provides administrative credentials only if 'admin' is
# specified as the username.
#  (string value)
# Possible values:
# keystone - <No description provided>
# noauth2 - <No description provided>
#auth_strategy = keystone


[cors]

#
# From oslo.middleware.cors
#

# Indicate whether this resource may be shared with the domain received in the
# requests "origin" header. Format: "<protocol>://<host>[:<port>]", no␣
↪trailing
# slash. Example: https://horizon.example.com (list value)
#allowed_origin = <None>

# Indicate that the actual request can include user credentials (boolean␣
↪value)
#allow_credentials = true
```

(continues on next page)

(continued from previous page)

```
# Indicate which headers are safe to expose to the API. Defaults to HTTP␣
↪Simple
# Headers. (list value)
#expose_headers =

# Maximum cache age of CORS preflight requests. (integer value)
#max_age = 3600

# Indicate which methods can be used during the actual request. (list value)
#allow_methods = OPTIONS,GET,HEAD,POST,PUT,DELETE,TRACE,PATCH

# Indicate which header field names may be used during the actual request.␣
↪(list
# value)
#allow_headers =


[keystone_authtoken]

#
# From keystonemiddleware.auth_token
#

# Complete "public" Identity API endpoint. This endpoint should not be an
# "admin" endpoint, as it should be accessible by all end users.␣
↪Unauthenticated
# clients are redirected to this endpoint to authenticate. Although this
# endpoint should ideally be unversioned, client support in the wild varies.␣
↪If
# you're using a versioned v2 endpoint here, then this should *not* be the same
# endpoint the service user utilizes for validating tokens, because normal end
# users may not be able to reach that endpoint. (string value)
# Deprecated group/name - [keystone_authtoken]/auth_uri
#www_authenticate_uri = <None>

# DEPRECATED: Complete "public" Identity API endpoint. This endpoint should␣
↪not
# be an "admin" endpoint, as it should be accessible by all end users.
# Unauthenticated clients are redirected to this endpoint to authenticate.
# Although this endpoint should ideally be unversioned, client support in the
# wild varies. If you're using a versioned v2 endpoint here, then this should
# *not* be the same endpoint the service user utilizes for validating tokens,
# because normal end users may not be able to reach that endpoint. This option
# is deprecated in favor of www_authenticate_uri and will be removed in the S
# release. (string value)
# This option is deprecated for removal since Queens.
# Its value may be silently ignored in the future.
# Reason: The auth_uri option is deprecated in favor of www_authenticate_uri␣
↪and
```

(continues on next page)

(continued from previous page)

```
# will be removed in the S  release.
#auth_uri = <None>

# API version of the Identity API endpoint. (string value)
#auth_version = <None>

# Interface to use for the Identity API endpoint. Valid values are "public",
# "internal" (default) or "admin". (string value)
#interface = internal

# Do not handle authorization requests within the middleware, but delegate the
# authorization decision to downstream WSGI components. (boolean value)
#delay_auth_decision = false

# Request timeout value for communicating with Identity API server. (integer
# value)
#http_connect_timeout = <None>

# How many times are we trying to reconnect when communicating with Identity␣
↪API
# Server. (integer value)
#http_request_max_retries = 3

# Request environment key where the Swift cache object is stored. When
# auth_token middleware is deployed with a Swift cache, use this option to␣
↪have
# the middleware share a caching backend with swift. Otherwise, use the
# ``memcached_servers`` option instead. (string value)
#cache = <None>

# Required if identity server requires client certificate (string value)
#certfile = <None>

# Required if identity server requires client certificate (string value)
#keyfile = <None>

# A PEM encoded Certificate Authority to use when verifying HTTPs connections.
# Defaults to system CAs. (string value)
#cafile = <None>

# Verify HTTPS connections. (boolean value)
#insecure = false

# The region in which the identity server can be found. (string value)
#region_name = <None>

# Optionally specify a list of memcached server(s) to use for caching. If left
# undefined, tokens will instead be cached in-process. (list value)
# Deprecated group/name - [keystone_authtoken]/memcache_servers
```

(continues on next page)

(continued from previous page)

```
#memcached_servers = <None>

# In order to prevent excessive effort spent validating tokens, the middleware
# caches previously-seen tokens for a configurable duration (in seconds). Set␣
↪to
# -1 to disable caching completely. (integer value)
#token_cache_time = 300

# (Optional) If defined, indicate whether token data should be authenticated␣
↪or
# authenticated and encrypted. If MAC, token data is authenticated (with HMAC)
# in the cache. If ENCRYPT, token data is encrypted and authenticated in the
# cache. If the value is not one of these options or empty, auth_token will
# raise an exception on initialization. (string value)
# Possible values:
# None - <No description provided>
# MAC - <No description provided>
# ENCRYPT - <No description provided>
#memcache_security_strategy = None

# (Optional, mandatory if memcache_security_strategy is defined) This string␣
↪is
# used for key derivation. (string value)
#memcache_secret_key = <None>

# (Optional) Number of seconds memcached server is considered dead before it␣
↪is
# tried again. (integer value)
#memcache_pool_dead_retry = 300

# (Optional) Maximum total number of open connections to every memcached␣
↪server.
# (integer value)
#memcache_pool_maxsize = 10

# (Optional) Socket timeout in seconds for communicating with a memcached
# server. (integer value)
#memcache_pool_socket_timeout = 3

# (Optional) Number of seconds a connection to memcached is held unused in the
# pool before it is closed. (integer value)
#memcache_pool_unused_timeout = 60

# (Optional) Number of seconds that an operation will wait to get a memcached
# client connection from the pool. (integer value)
#memcache_pool_conn_get_timeout = 10

# (Optional) Use the advanced (eventlet safe) memcached client pool. (boolean
# value)
```

(continues on next page)

(continued from previous page)

```
#memcache_use_advanced_pool = true

# (Optional) Indicate whether to set the X-Service-Catalog header. If False,
# middleware will not ask for service catalog on token validation and will not
# set the X-Service-Catalog header. (boolean value)
#include_service_catalog = true

# Used to control the use and type of token binding. Can be set to: "disabled"
# to not check token binding. "permissive" (default) to validate binding
# information if the bind type is of a form known to the server and ignore it␣
↪if
# not. "strict" like "permissive" but if the bind type is unknown the token␣
↪will
# be rejected. "required" any form of token binding is needed to be allowed.
# Finally the name of a binding method that must be present in tokens. (string
# value)
#enforce_token_bind = permissive

# A choice of roles that must be present in a service token. Service tokens␣
↪are
# allowed to request that an expired token can be used and so this check␣
↪should
# tightly control that only actual services should be sending this token.␣
↪Roles
# here are applied as an ANY check so any role in this list must be present.␣
↪For
# backwards compatibility reasons this currently only affects the allow_
↪expired
# check. (list value)
#service_token_roles = service

# For backwards compatibility reasons we must let valid service tokens pass␣
↪that
# don't pass the service_token_roles check as valid. Setting this true will
# become the default in a future release and should be enabled if possible.
# (boolean value)
#service_token_roles_required = false

# The name or type of the service as it appears in the service catalog. This␣
↪is
# used to validate tokens that have restricted access rules. (string value)
#service_type = <None>

# Authentication type to load (string value)
# Deprecated group/name - [keystone_authtoken]/auth_plugin
#auth_type = <None>

# Config Section from which to load plugin specific options (string value)
#auth_section = <None>
```

(continues on next page)

(continued from previous page)

```
[oslo_policy]

#
# From oslo.policy
#

# This option controls whether or not to enforce scope when evaluating␣
↪policies.
# If ``True``, the scope of the token used in the request is compared to the
# ``scope_types`` of the policy being enforced. If the scopes do not match, an
# ``InvalidScope`` exception will be raised. If ``False``, a message will be
# logged informing operators that policies are being invoked with mismatching
# scope. (boolean value)
#enforce_scope = false

# This option controls whether or not to use old deprecated defaults when
# evaluating policies. If ``True``, the old deprecated defaults are not going to
# be evaluated. This means if any existing token is allowed for old defaults␣
↪but
# is disallowed for new defaults, it will be disallowed. It is encouraged to
# enable this flag along with the ``enforce_scope`` flag so that you can get the
# benefits of new defaults and ``scope_type`` together (boolean value)
#enforce_new_defaults = false

# The relative or absolute path of a file that maps roles to permissions for a
# given service. Relative paths must be specified in relation to the
# configuration file setting this option. (string value)
#policy_file = policy.json

# Default rule. Enforced when a requested rule is not found. (string value)
#policy_default_rule = default

# Directories where policy configuration files are stored. They can be␣
↪relative
# to any directory in the search path defined by the config_dir option, or
# absolute paths. The file defined by policy_file must exist for these
# directories to be searched.  Missing or empty directories are ignored.␣
↪(multi
# valued)
#policy_dirs = policy.d

# Content Type to send and receive data for REST based policy check (string
# value)
# Possible values:
# application/x-www-form-urlencoded - <No description provided>
# application/json - <No description provided>
#remote_content_type = application/x-www-form-urlencoded
```

(continues on next page)

(continued from previous page)

```
# server identity verification for REST based policy check (boolean value)
#remote_ssl_verify_server_crt = false

# Absolute path to ca cert file for REST based policy check (string value)
#remote_ssl_ca_crt_file = <None>

# Absolute path to client cert for REST based policy check (string value)
#remote_ssl_client_crt_file = <None>

# Absolute path client key file REST based policy check (string value)
#remote_ssl_client_key_file = <None>


[placement]

#
# From placement.conf
#

#
# If True, when limiting allocation candidate results, the results will be
# a random sampling of the full result set. If False, allocation candidates
# are returned in a deterministic but undefined order. That is, all things
# being equal, two requests for allocation candidates will return the same
# results in the same order; but no guarantees are made as to how that order
# is determined.
#  (boolean value)
#randomize_allocation_candidates = false

#
# Early API microversions (<1.8) allowed creating allocations and not␣
↪specifying
# a project or user identifier for the consumer. In cleaning up the data
# modeling, we no longer allow missing project and user information. If an␣
↪older
# client makes an allocation, we'll use this in place of the information it
# doesn't provide.
#  (string value)
#incomplete_consumer_project_id = 00000000-0000-0000-0000-000000000000

#
# Early API microversions (<1.8) allowed creating allocations and not␣
↪specifying
# a project or user identifier for the consumer. In cleaning up the data
# modeling, we no longer allow missing project and user information. If an␣
↪older
# client makes an allocation, we'll use this in place of the information it
# doesn't provide.
```

(continues on next page)

(continued from previous page)

```
#  (string value)
#incomplete_consumer_user_id = 00000000-0000-0000-0000-000000000000

#
# The number of times to retry, server-side, writing allocations when there is
# a resource provider generation conflict. Raising this value may be useful
# when many concurrent allocations to the same resource provider are expected.
#  (integer value)
#allocation_conflict_retry_count = 10


[placement_database]
#
# The *Placement API Database* is a the database used with the placement
# service. If the connection option is not set, the placement service will
# not start.

#
# From placement.conf
#

# The SQLAlchemy connection string to use to connect to the database. (string
# value)
#connection = <None>

# Optional URL parameters to append onto the connection URL at connect time;
# specify as param1=value1&param2=value2&... (string value)
#connection_parameters =

# If True, SQLite uses synchronous mode. (boolean value)
#sqlite_synchronous = true

# The SQLAlchemy connection string to use to connect to the slave database.
# (string value)
#slave_connection = <None>

# The SQL mode to be used for MySQL sessions. This option, including the
# default, overrides any server-set SQL mode. To use whatever SQL mode is set␣
↪by
# the server configuration, set this to no value. Example: mysql_sql_mode=
# (string value)
#mysql_sql_mode = TRADITIONAL

# Connections which have been present in the connection pool longer than this
# number of seconds will be replaced with a new one the next time they are
# checked out from the pool. (integer value)
#connection_recycle_time = 3600

# Maximum number of SQL connections to keep open in a pool. Setting a value␣
↪of 0
```

(continues on next page)

(continued from previous page)

```
# indicates no limit. (integer value)
#max_pool_size = <None>

# Maximum number of database connection retries during startup. Set to -1 to
# specify an infinite retry count. (integer value)
#max_retries = 10

# Interval between retries of opening a SQL connection. (integer value)
#retry_interval = 10

# If set, use this value for max_overflow with SQLAlchemy. (integer value)
#max_overflow = <None>

# Verbosity of SQL debugging information: 0=None, 100=Everything. (integer
# value)
#connection_debug = 0

# Add Python stack traces to SQL as comment strings. (boolean value)
#connection_trace = false

# If set, use this value for pool_timeout with SQLAlchemy. (integer value)
#pool_timeout = <None>

# If True, database schema migrations will be attempted when the web service
# starts. (boolean value)
#sync_on_startup = false


[profiler]

#
# From osprofiler
#


#
# Enable the profiling for all services on this node.
#
# Default value is False (fully disable the profiling feature).
#
# Possible values:
#
# * True: Enables the feature
# * False: Disables the feature. The profiling cannot be started via this
# project
#   operations. If the profiling is triggered by another project, this project
#   part will be empty.
#  (boolean value)
# Deprecated group/name - [profiler]/profiler_enabled
#enabled = false
```

(continues on next page)

(continued from previous page)

```
#
# Enable SQL requests profiling in services.
#
# Default value is False (SQL requests won't be traced).
#
# Possible values:
#
# * True: Enables SQL requests profiling. Each SQL query will be part of the
#   trace and can the be analyzed by how much time was spent for that.
# * False: Disables SQL requests profiling. The spent time is only shown on a
#   higher level of operations. Single SQL queries cannot be analyzed this␣
↪way.
#  (boolean value)
#trace_sqlalchemy = false

#
# Secret key(s) to use for encrypting context data for performance profiling.
#
# This string value should have the following format: <key1>[,<key2>,...<keyn>
↪],
# where each key is some random string. A user who triggers the profiling via
# the REST API has to set one of these keys in the headers of the REST API␣
↪call
# to include profiling results of this node for this particular project.
#
# Both "enabled" flag and "hmac_keys" config options should be set to enable
# profiling. Also, to generate correct profiling information across all␣
↪services
# at least one key needs to be consistent between OpenStack projects. This
# ensures it can be used from client side to generate the trace, containing
# information from all possible resources.
#  (string value)
#hmac_keys = SECRET_KEY

#
# Connection string for a notifier backend.
#
# Default value is ``messaging://`` which sets the notifier to oslo_messaging.
#
# Examples of possible values:
#
# * ``messaging://`` - use oslo_messaging driver for sending spans.
# * ``redis://127.0.0.1:6379`` - use redis driver for sending spans.
# * ``mongodb://127.0.0.1:27017`` - use mongodb driver for sending spans.
# * ``elasticsearch://127.0.0.1:9200`` - use elasticsearch driver for sending
#   spans.
# * ``jaeger://127.0.0.1:6831`` - use jaeger tracing as driver for sending
# spans.
```

(continues on next page)

(continued from previous page)

```
#  (string value)
#connection_string = messaging://

#
# Document type for notification indexing in elasticsearch.
#  (string value)
#es_doc_type = notification

#
# This parameter is a time value parameter (for example: es_scroll_time=2m),
# indicating for how long the nodes that participate in the search will␣
↪maintain
# relevant resources in order to continue and support it.
#  (string value)
#es_scroll_time = 2m

#
# Elasticsearch splits large requests in batches. This parameter defines
# maximum size of each batch (for example: es_scroll_size=10000).
#  (integer value)
#es_scroll_size = 10000

#
# Redissentinel provides a timeout option on the connections.
# This parameter defines that timeout (for example: socket_timeout=0.1).
#  (floating point value)
#socket_timeout = 0.1

#
# Redissentinel uses a service name to identify a master redis service.
# This parameter defines the name (for example:
# ``sentinal_service_name=mymaster``).
#  (string value)
#sentinel_service_name = mymaster

#
# Enable filter traces that contain error/exception to a separated place.
#
# Default value is set to False.
#
# Possible values:
#
# * True: Enable filter traces that contain error/exception.
# * False: Disable the filter.
#  (boolean value)
#filter_error_trace = false
```

CHAPTER

FOUR

CONTRIBUTION

4.1 Placement Developer Notes

The Nova project introduced the placement service as part of the Newton release, and it was extracted to its own repository in the Stein release. The service provides an HTTP API to manage inventories of different classes of resources, such as disk or virtual cpus, made available by entities called resource providers. Information provided through the placement API is intended to enable more effective accounting of resources in an OpenStack deployment and better scheduling of various entities in the cloud.

The document serves to explain the architecture of the system and to provide some guidance on how to maintain and extend the code. For more detail on why the system was created and how it does its job see *Placement*. For some insight into the longer term goals of the system see *Goals* and *Vision Reflection*.

4.1.1 So You Want to Contribute...

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with placement.

Communication

As an official OpenStack project, Placement follows the overarching processes outlined in the Project Team Guide. Contribution is welcomed from any interested parties and takes many different forms.

To make sure everything gets the attention it deserves and work is not duplicated there are some guidelines, stated here.

If in doubt, ask someone, either by sending a message to the openstack-discuss mailing list with a `[placement]` subject tag, or by visiting the `#openstack-nova` IRC channel on `webchat.oftc.net`.

StoryBoard

Placement uses StoryBoard for tracking bugs, features, "cleanups" (described below), and docs improvements. Different types of stories are distinguished by tags which are applied manually by people creating or reviewing the stories. These tags are used to create work lists.

Table 1: Worklists

List	Main Tag	Description
Untriaged Stories	No tags	New or unvisited stories which have not yet been categorized (and should be).
Bugs	`bug`	Incorrect behavior in existing code, features or documents. If the issue is with documentation, add the tag `docs`.
Features	`rfe`	Planned or requested features or enhancements to the system.
Cleanups	`cleanup`	Improvements to the code or surrounding repository to help with maintenance or other development lifecycle issues that are not otherwise a bug or feature.
Docs	`docs`	Required or recommended improvements to the documentation.

When all the tasks in a story have a status of `Merged` or `Invalid`, the story will be considered complete and its status will change from `Active` to either `Merged` or `Invalid`. If there is at least one task in the `Merged` state, the story will be `Merged`. The worklists above are configured to only show stories that are `Active`.

Note: These worklists and their chosen tags are subject to change as the project gains more experience with StoryBoard. The tags chosen thus far are, in part, the result of them already existing elsewhere in the system.

Submitting and Managing Bugs

Bugs found in placement should be reported in StoryBoard by creating a new story in the `openstack/placement` project.

New Bugs

If you are submitting a new bug, explain the problem, the steps taken that led to the bad results, and the expected results. Please also add as much of the following information as possible:

- Relevant lines from the `placement-api` log.
- The OpenStack release (e.g., `Stein`).
- The method used to install or deploy placement.
- The operating system(s) on which the placement is running.
- The version(s) of Python being used.

Tag the story with `bug`.

Triaging Bugs

Triaging newly submitted bugs to confirm they really are bugs, gather missing information, and to suggest possible solutions is one of the most important contributions that can be made to any open source project.

If a new bug doesn't have the `bug` tag, add it. If it isn't a functional problem with existing code, consider adding the `rfe` tag for a feature request, or `cleanup` for a suggested refactoring or a reminder for future work. An error in documentation is a `bug`.

Leave comments on the story if you have questions or ideas. If you are relatively certain about a solution, add the steps of that solution as tasks on the story.

If submitting a change related to a story, the gerrit system will automatically link to StoryBoard if you include `Story:` and, optionally, `Task:` identifiers in your commit message, like this:

```
Story: 2005190
Task: 29938
```

Using solely `Story:` will leave a comment referring back to the commit in gerrit. Adding `Task:` will update the identified task to indicate it is in a `Review` state. When the change merges, the state will be updated to `Merged`.

Reviewing Code

Like other OpenStack projects, Placement uses gerrit to facilitate peer code review. It is hoped and expected that anyone who would like to commit code to the Placement project will also spend time reviewing code for the sake of the common good. The more people reviewing, the more code that will eventually merge.

See How to Review Changes the OpenStack Way for an overview of the review and voting process.

There is a small group of people within the Placement team called core reviewers. These are people who are empowered to signal (via the +2 vote) that code is of a suitable standard to be merged and is aligned with the current goals of the project. Core reviewers are regularly selected from all active reviewers based on the quantity and quality of their reviews and demonstrated understanding of the Placement code and goals of the project.

The point of review is to evolve potentially useful code to merged working code that is aligned with the standards of style, testing, and correctness that we share as group. It is not for creating perfect code. Review should always be constructive, encouraging, and friendly. People who contribute code are doing the project a favor, make it feel that way.

Some guidelines that reviewers and patch submitters should be aware of:

- It is very important that a new patch set gets some form of review as soon as possible, even if only to say "we've seen this". Latency in the review process has been identified as hugely discouraging for new and experienced contributors alike.
- Follow up changes, to fix minor problems identified during review, are encouraged. We want to keep things moving.
- As a reviewer, remember that not all patch submitters will know these guidelines. If it seems they don't, point them here and be patient in the meantime.

- Gerrit can be good for code review, but is often not a great environment for having a discussion that is struggling to resolve to a decision. Move discussion to the mailing list sooner rather than later. Add a link to the thread in the list archive to the review.
- If the CI system is throwing up random failures in test runs, you should endeavor whenever possible to investigate, not simply `recheck`. A flakey gate is an indication that OpenStack is not robust and at the root of all this, making OpenStack work well is what we are doing.

Special Considerations For Core Reviewers

Core reviewers have special powers. With great power comes great responsibility and thus being held to a standard. As a core reviewer, your job is to enable other people to contribute good code. Under ideal conditions it is more important to be reviewing other people's code and bugs and fixing bugs than it is to be writing your own features. Frequently conditions will not be ideal, but strive to enable others.

When there are open questions that need to be resolved, try to prefer the openstack-discuss list over IRC so that anyone can be involved according to their own schedules and input from unexpected sources can be available.

Writing Code

This document cannot enumerate all the many ways to write good Python code. Instead it lists some guidelines that, if followed, will help make sure your code is reviewed promptly and merges quickly. As with everything else in this document, these guidelines will evolve over time and may be violated for special circumstances. If you have questions, ask.

See *Placement Developer Notes* for an overview of Placement and how the various pieces fit together.

- Divide your change into a series of commits each of which encapsulates a single unit of functionality but still results in a working service. Smaller changes are easier to review.
- If your change is to the HTTP API, familiarize yourself with *Microversions*.
- If there is a series of changes leading to an HTTP API change, exposing that API change should be the last patch in the series. That patch must update the API reference and include a release note.
- Changes must include tests. There is a separate document on *Testing Placement*.
- Run `tox` before submitting your code to gerrit. This will run unit and functional tests in both Python 2 and Python 3, and pep8 style checks. Placement tests, including functional, are fast, so this should not be too much of a hardship. By running the tests locally you avoid wasting scarce resources in the CI system.
- Keep the tests fast. Avoid sleeps, network connections, and external processes in the tests.
- Keep Placement fast. There is a `placement-perfload` job that runs with every patch. Within that is a log file, `/logs/placement-perf.txt[.gz]` that gives rough timing information for a common operation. We want those numbers to stay small.
- We follow the code formatting guidelines of PEP 8. Check your code with `tox -epep8` (for all files) or `tox -efast8` (for just the files you changed). You will not always agree with the advice provided. Follow it.
- Where possible avoid using the visual indent style. Using it can make future changes unnecessarily difficult. This guideline is not enforced by pep8 and has been used throughout the code in the past. There's no need to fix old use. Instead of this

```
return_value = self.some_method(arg1, arg2,
                                arg3, arg4)
```

prefer this

```
return_value = self.some_method(
    arg1, arg2, arg3, arg4)
```

- Changes associated with stories and tasks in StoryBoard should include `Story` and `Task` identifiers in the commit message, as described in *Triaging Bugs* above.

New Features

New functionality in Placement is developed as needed to meet new use cases or improve the handling of existing use cases. As a service used by other services in OpenStack, uses cases often originate in those other services. Considerable collaboration with other projects is often required to determine if any changes are needed in the Placement API or elsewhere in the project. That interaction should happen in the usual ways: At Project Team Gatherings, on the openstack-discuss list, and in IRC.

Once there is a clear need for a change, a story should be created in StoryBoard with a tag of `rfe`. Placement team members will evaluate the story to determine if a *spec* is required. If it is, a task to create the spec will be added to the story. At this time there are no hard and fast rules on what will require a spec. If the implementation is well understood it may be the case that a detailed story and a series of tasks associated with that story will be sufficient. If further discussion is required to understand the problem or to evolve or verify the design of the solution, a spec is a good idea.

If a spec is required there are some guidelines for creating one:

- A file should be created in the placement code in `doc/source/specs/<cycle-name>/approved` with a filename beginning with the identifier of the story. For example:

  ```
  docs/source/specs/train/approved/200056-infinite-resource-classes.rst
  ```

  More details on how to write a spec are included in a `template.rst` file found in the `doc/source/specs` directory. This may be copied to use as the starting point for a new spec.

- Under normal circumstances specs should be proposed near the beginning of a release cycle so there is sufficient time to review the spec and its implementation as well as to make any necessary decisions about limiting the number of specs being worked in the same cycle. Unless otherwise announced at the beginning of a cycle, specs should merge before milestone-2 to be considered relevant for that cycle. Exceptions will be reviewed on a case by case basis. See the stein schedule for an example schedule.

- Work items that are described in a spec should be reflected as tasks created on the originating story. Update the story with additional tasks as they are discovered. Most new tasks will not require updating the spec.

- If, when developing a feature, the implementation significantly diverges from the spec, the spec should be updated to reflect the new reality. This should not be considered exceptional: It is normal for there to be learning during the development process which impacts the solution.

- Though specs are presented with the Placement documentation and can usefully augment end-user documentation, they are not a substitute. Development of a new feature is not complete without documentation.

When a spec was approved in a previous release cycle, but was not finished, it should be re-proposed (via gerrit) to the current cycle. Include `Previously-Approved: <cycle>` in the commit message to highlight that fact. If there have been no changes, core reviewers should feel free to fast-approve (only one +2 required) the change.

Project Team Lead Duties

PTL duties are enumerated in the PTL guide.

4.1.2 Architecture

The placement service is straightforward: It is a WSGI application that sends and receives JSON, using an RDBMS (usually MySQL) for persistence. As state is managed solely in the DB, scaling the placement service is done by increasing the number of WSGI application instances and scaling the RDBMS using traditional database scaling techniques.

For sake of consistency and because there was initially intent to make the entities in the placement service available over RPC, versioned objects were used to provide the interface between the HTTP application layer and the SQLAlchemy-driven persistence layer. In the Stein release, that interface was refactored to remove the use of versioned objects and split functionality into smaller modules.

Though the placement service does not aspire to be a *microservice* it does aspire to continue to be small and minimally complex. This means a relatively small amount of middleware that is not configurable, and a limited number of exposed resources where any given resource is represented by one (and only one) URL that expresses a noun that is a member of the system. Adding additional resources should be considered a significant change requiring robust review from many stakeholders.

The set of HTTP resources represents a concise and constrained grammar for expressing the management of resource providers, inventories, resource classes, traits, and allocations. If a solution is initially designed to need more resources or a more complex grammar that may be a sign that we need to give our goals greater scrutiny. Is there a way to do what we want with what we have already? Can some other service help? Is a new collaborating service required?

Minimal Framework

The API is set up to use a minimal framework that tries to keep the structure of the application as discoverable as possible and keeps the HTTP interaction near the surface. The goal of this is to make things easy to trace when debugging or adding functionality.

Functionality which is required for every request is handled in raw WSGI middleware that is composed in the *placement.deploy* module. Dispatch or routing is handled declaratively via the `ROUTE_DECLARATIONS` map defined in the *placement.handler* module.

Mapping is by URL plus request method. The destination is a complete WSGI application, using a subclass of the wsgify method from WebOb to provide a Request object that provides convenience methods for accessing request headers, bodies, and query parameters and for generating responses. In the placement API these mini-applications are called *handlers*. The *wsgify* subclass is provided in *placement.wsgi_wrapper* as *PlacementWsgify*. It is used to make sure that JSON formatted error responses are structured according to the API-SIG errors guideline.

This division between middleware, dispatch and handlers is supposed to provide clues on where a particular behavior or functionality should be implemented. Like most such systems, this does not always work but is a useful tool.

Microversions

The placement API makes use of microversions to allow the release of new features on an opt in basis. See *Placement* for an up to date history of the available microversions.

The rules around when a microversion is needed are modeled after those of the compute API. When adding a new microversion there are a few bits of required housekeeping that must be done in the code:

- Update the `VERSIONS` list in `placement/microversion.py` to indicate the new microversion and give a very brief summary of the added feature.
- Update `placement/rest_api_version_history.rst` to add a more detailed section describing the new microversion.
- Add a release note with a `features` section announcing the new or changed feature and the microversion.
- If the `version_handler` decorator (see below) has been used, increment `TOTAL_VERSIONED_METHODS` in `placement/tests/unit/test_microversion.py`. This provides a confirmatory check just to make sure you are paying attention and as a helpful reminder to do the other things in this list.
- Include functional gabbi tests as appropriate (see *Testing Placement*). At the least, update the `latest microversion` test in `placement/tests/functional/gabbits/microversion.yaml`.
- Update the API Reference documentation as appropriate. The source is located under `api-ref/source/`.
- If a new error code has been added in `placement/errors.py`, it should be added to the API Reference.

In the placement API, microversions only use the modern form of the version header:

```
OpenStack-API-Version: placement 1.2
```

If a valid microversion is present in a request it will be placed, as a `Version` object, into the WSGI environment with the `placement.microversion` key. Often, accessing this in handler code directly (to control branching) is the most explicit and granular way to have different behavior per microversion. A `Version` instance can be treated as a tuple of two ints and compared as such or there is a `matches` method.

A `version_handler` decorator is also available. It makes it possible to have multiple different handler methods of the same (fully-qualified by package) name, each available for a different microversion window. If a request wants a microversion that is not available, a defined status code is returned (usually `404` or `405`). There is a unit test in place which will fail if there are version intersections.

Adding a New Handler

Adding a new URL or a new method (e.g, `PATCH`) to an existing URL requires adding a new handler function. In either case a new microversion and release note is required. When adding an entirely new route a request for a lower microversion should return a `404`. When adding a new method to an existing URL a request for a lower microversion should return a `405`.

In either case, the `ROUTE_DECLARATIONS` dictionary in the *placement.handler* module should be updated to point to a function within a module that contains handlers for the type of entity identified by the URL. Collection and individual entity handlers of the same type should be in the same module.

As mentioned above, the handler function should be decorated with `@wsgi_wrapper.PlacementWsgify`, take a single argument `req` which is a WebOb Request object, and return a WebOb Response.

For `PUT` and `POST` methods, request bodies are expected to be JSON based on a content-type of `application/json`. This may be enforced by using a decorator: `@util.require_content('application/json')`. If the body is not *JSON*, a `415` response status is returned.

Response bodies are usually *JSON*. A handler can check the *Accept* header provided in a request using another decorator: `@util.check_accept('application/json')`. If the header does not allow *JSON*, a `406` response status is returned.

If a hander returns a response body, a `Last-Modified` header should be included with the response. If the entity or entities in the response body are directly associated with an object (or objects, in the case of a collection response) that has an `updated_at` (or `created_at`) field, that field's value can be used as the value of the header (WebOb will take care of turning the datetime object into a string timestamp). A `util.pick_last_modified` is available to help choose the most recent last-modified when traversing a collection of entities.

If there is no directly associated object (for example, the output is the composite of several objects) then the `Last-Modified` time should be `timeutils.utcnow(with_timezone=True)` (the timezone must be set in order to be a valid HTTP timestamp). For example, the response to `GET /allocation_candidates` should have a last-modified header of now because it is composed from queries against many different database entities, presents a mixture of result types (allocation requests and provider summaries), and has a view of the system that is only meaningful *now*.

If a `Last-Modified` header is set, then a `Cache-Control` header with a value of `no-cache` must be set as well. This is to avoid user-agents inadvertently caching the responses.

JSON sent in a request should be validated against a JSON Schema. A `util.extract_json` method is available. This takes a request body and a schema. If multiple schema are used for different microversions of the same request, the caller is responsible for selecting the right one before calling `extract_json`.

When a handler needs to read or write the data store it should use methods on the objects found in the *placement.objects* package. Doing so requires a context which is provided to the handler method via the WSGI environment. It can be retrieved as follows:

```
context = req.environ['placement.context']
```

Note: If your change requires new methods or new objects in the *placement.objects* package, after you have made sure that you really do need those new methods or objects (you may not!) make those changes in a patch that is separate from and prior to the HTTP API change.

If a handler needs to return an error response, with the advent of Placement API Error Handling, it is possible to include a code in the JSON error response. This can be used to distinguish different errors with the same HTTP response status code (a common case is a generation conflict versus an inventory in use conflict). Error codes are simple namespaced strings (e.g., `placement.inventory.inuse`) for which symbols are maintained in `placement.errors`. Adding a symbol to a response is done by using the `comment` kwarg to a WebOb exception, like this:

```
except exception.InventoryInUse as exc:
    raise webob.exc.HTTPConflict(
        _('update conflict: %(error)s') % {'error': exc},
        comment=errors.INVENTORY_INUSE)
```

Code that adds newly raised exceptions should include an error code. Find additional guidelines on use in the docs for `placement.errors`. When a new error code is added, also document it in the API Reference.

Testing of handler code is described in *Testing Placement*.

Database Schema Changes

At some point in every application's life it becomes necessary to change the structure of its database. Modifying the SQLAlchemy models (in placement/db/sqlachemy/models.py) is necessary for the application to understand the new structure, but that will not change the actual underlying database. To do that, Placement uses *alembic* to run database migrations.

Alembic calls each change a **revision**. To create a migration with alembic, run the *alembic revision* command. Alembic will then generate a new revision file with a unique file name, and place it in the *alembic/versions/* directory:

```
ed@devenv:~/projects/placement$ alembic -c placement/db/sqlalchemy/alembic.
↪ini revision -m "Add column foo to bar table"
Generating /home/ed/projects/placement/placement/db/sqlalchemy/alembic/
↪versions/dfb006498ad2_add_column_foo_to_bar_table.py ... done
```

Let us break down that command:

- The **-c** parameter tells alembic where to find its configuration file.
- **revision** is the alembic subcommand for creating a new revision file.
- The **-m** parameter specifies a brief comment explaining the change.
- The generated file from alembic will have a name consisting of a random hash prefix, followed by an underscore, followed by your **-m** comment, and a **.py** extension. So be sure to keep your comment as brief as possible while still being descriptive.

The generated file will look something like this:

```
"""Add column foo to bar table

Revision ID: dfb006498ad2
Revises: 0378df171af3
Create Date: 2018-10-29 20:02:58.290779

```

(continues on next page)

(continued from previous page)

```
"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = 'dfb006498ad2'
down_revision = '0378df171af3'
branch_labels = None
depends_on = None


def upgrade():
    pass
```

The top of the file is the docstring that will show when you review your revision history. If we did not include the **-m** comment when we ran the *alembic revision* command, this would just contain "empty message". If you did not specify the comment when creating the file, be sure to replace "empty message" with a brief comment describing the reason for the database change.

You then need to define the changes in the *upgrade()* method. The code used in these methods is basic SQLAlchemy code for creating and modifying tables. You can examine existing migrations in the project to see examples of what this code looks like, as well as find more in-depth usage of Alembic in the Alembic tutorial.

One other option when creating the revision is to add the `--autogenerate` parameter to the revision command. This assumes that you have already updated the SQLAlchemy models, and have a connection to the placement database configured. When run with this option, the *upgrade()* method of the revision file is filled in for you by alembic as it compares the schema described in your models.py script and the actual state of the database. You should always verify the revision script to make sure it does just what you intended, both by reading the code as well as running the tests, as there are some things that autogenerate cannot deduce. See autogenerate limitations for more detailed information.

Gotchas

This section tries to shed some light on some of the differences between the placement API and some of the other OpenStack APIs or on situations which may be surprising or unexpected.

- The placement API is somewhat more strict about *Content-Type* and *Accept* headers in an effort to follow the HTTP RFCs.

  If a user-agent sends some JSON in a *PUT* or *POST* request without a *Content-Type* of *application/json* the request will result in an error.

  If a *GET* request is made without an *Accept* header, the response will default to being *application/json*.

  If a request is made with an explicit *Accept* header that does not include *application/json* then there will be an error and the error will attempt to be in the requested format (for example, *text/plain*).

- If a URL exists, but a request is made using a method that that URL does not support, the API will respond with a *405* error. Sometimes in the nova APIs this can be a *404* (which is wrong, but understandable given the constraints of the code).

- Because each handler is individually wrapped by the *PlacementWsgify* decorator any exception that is a subclass of *webob.exc.WSGIHTTPException* that is raised from within the handler, such as *webob.exc.HTTPBadRequest*, will be caught by WebOb and turned into a valid Response containing headers and body set by WebOb based on the information given when the exception was raised. It will not be seen as an exception by any of the middleware in the placement stack.

  In general this is a good thing, but it can lead to some confusion if, for example, you are trying to add some middleware that operates on exceptions.

  Other exceptions that are not from WebOb will raise outside the handlers where they will either be caught in the *__call__* method of the *PlacementHandler* app that is responsible for dispatch, or by the *FaultWrap* middleware.

4.1.3 API reference guideline

The API reference should be updated when placement APIs are modified (microversion is bumped, etc.). This page describes the guideline for updating the API reference.

API reference

- Placement API reference

The guideline to write the API reference

The API reference consists of the following files.

- API reference text: `api-ref/source/*.inc`
- Parameter definition: `api-ref/source/parameters.yaml`
- JSON request/response samples: `api-ref/source/samples/*`

Structure of inc file

Each REST API is described in the text file (*.inc). The structure of inc file is as follows:

- Title
  - API Name
    - REST Method
      - URL
      - Description
      - Normal status code
      - Error status code
      - Request
        - Parameters
        - JSON request body example (if exists)

        - Response
            - Parameters
            - JSON response body example (if exists)
    - API Name (Next)
        * ...

REST Method

The guideline for describing HTTP methods is described in this section. All supported methods by resource should be listed in the API reference.

The order of methods

Methods have to be sorted by each URI in the following order:

1. GET
2. POST
3. PUT
4. PATCH (unused by Nova)
5. DELETE

And sorted from broadest to narrowest. So for /severs it would be:

1. GET /servers
2. POST /servers
3. GET /servers/details
4. GET /servers/{server_id}
5. PUT /servers/{server_id}
6. DELETE /servers/{server_id}

Method titles spelling and case

The spelling and the case of method names in the title have to match what is in the code. For instance, the title for the section on method "Get Rdp Console" should be "Get Rdp Console (os-getRDPConsole Action)" NOT "Get Rdp Console (Os-Getrdpconsole Action)"

Response codes

The normal response codes (20x) and error response codes have to be listed. The order of response codes should be in ascending order. The description of typical error response codes are as follows:

Table 2: Error response codes

Response codes	Description
400	badRequest(400)
401	unauthorized(401)
403	forbidden(403)
404	itemNotFound(404)
409	conflict(409)
410	gone(410)
501	notImplemented(501)
503	serviceUnavailable(503)

Parameters

Parameters need to be defined by 2 subsections. The one is in the 'Request' subsection, the other is in the 'Response' subsection. The queries, request headers and attributes go in the 'Request' subsection and response headers and attributes go in the 'Response' subsection.

The order of parameters in each API

The request and response parameters have to be listed in the following order in each API in the text file.

1. Header
2. Path
3. Query
4. Body
    a. Top level object (i.e. server)
    b. Required fields
    c. Optional fields
    d. Parameters added in microversions (by the microversion they were added)

Parameter type

The parameters are defined in the parameter file (`parameters.yaml`). The type of parameters have to be one of followings:

- `array`

  It is a list.

- `boolean`

- `float`
- `integer`
- `none`

  The value is always `null` in a response or should be `null` in a request.

- `object`

  The value is dict.

- `string`

  If the value can be specified by multiple types, specify one type in the file and mention the other types in the description.

Required or optional

In the parameter file, define the `required` field in each parameter.

`true`	The parameter must be specified in the request, or the parameter always appears in the response.
`false`	It is not always necessary to specify the parameter in the request, or the parameter does not appear in the response in some cases. e.g. A config option defines whether the parameter appears in the response or not. A parameter appears when administrators call but does not appear when non-admin users call.

If a parameter must be specified in the request or always appears in the response in the micoversion added or later, the parameter must be defined as required (`true`).

The order of parameters in the parameter file

The order of parameters in the parameter file has to be kept as follows:

1. By in type
   a. Header
   b. Path
   c. Query
   d. Body
2. Then alphabetical by name

Example

Todo: The guideline for request/response JSON bodies should be added.

Body

Todo: The guideline for the introductory text and the context for the resource in question should be added.

Reference

- The description for Parameters whose values are null
- The definition of "Optional" parameter

4.1.4 Goals

Like many OpenStack projects, placement uses blueprints and specifications to plan and design upcoming work. Sometimes, however, certain types of work fit more in the category of wishlist, or when-we-get-around-to-it. These types of work are often not driven by user or operator feature requests, but are instead related to architectural, maintenance, and technical debt management goals that will make the lives of contributors to the project easier over time. In those cases a specification is too formal and detailed but it is still worthwhile to remember the idea and put it somewhere. That's what this document is for: a place to find and put goals for placement that are related to making contribution more pleasant and keep the project and product healthy, yet are too general to be considered feature requests.

This document can also operate as one of several sources of guidance on how not to stray too far from the long term vision of placement.

Don't Use Global Config

Placement uses oslo.config to manage configuration, passing a reference to an *oslo_config.cfg.ConfigOpts* as required. Before things were changed a global was used instead. Placement inherited this behavior from nova, where using a global `CONF` is the normal way to interact with the configuration options. Continuing this pattern in placement made it difficult for nova to use externalized placement in its functional tests, so the situation was changed. We'd like to keep it this way as it makes the code easier to maintain.

4.1.5 Quick Placement Development

Note: This is one of many ways to achieve a simple *live* development environment for the placement service. This isn't meant to be the best way, or the only way. Its purpose is more to demonstrate the steps involved, so that people can learn from those steps and choose to assemble them in whatever ways work best for them.

This content was originally written in a blog post, which perhaps explains its folksy tone.

Here are some instructions on how to spin up the placement wsgi script with uwsgi and a stubbed out `placement.conf`, in case you want to see what happens. The idea here is that you want to experiment with the current placement code, using a live database, but you're not concerned with other services, don't want to deal with devstack, but need a level of interaction with the code and process that something like placedock can't provide.

As ever, even all of the above has lots of assumptions about experience and context. This document assumes you are someone who either is an OpenStack (and probably placement) developer, or would like to be one.

To make this go you need a unix-like OS, with a python3 dev environment, and git and mysql (or postgresql) installed. We'll be doing this work from within a virtualenv, built from the `tox.ini` in the placement code.

Get The Code

The placement code lives at https://opendev.org/openstack/placement . We want to clone that:

```
git clone https://opendev.org/openstack/placement
cd placement
```

Setup The Database

We need to 1) create the database, 2) create a virtualenv to have the command, 3) use it to create the tables.

The database can have whatever name you like. Whatever you choose, use it throughout this process. We choose `placement`. You may need a user and password to talk to your database, setting that up is out of scope for this document:

```
mysql -uroot -psecret -e "DROP DATABASE IF EXISTS placement;"
mysql -uroot -psecret -e "CREATE DATABASE placement CHARACTER SET utf8;"
```

You may also need to set permissions:

```
mysql -uroot -psecret \
    -e "GRANT ALL PRIVILEGES ON placement.* TO 'root'@'%' identified by
↪'secret';"
```

Create a bare minimum placement.conf in the `/etc/placement` directory (which you may need to create):

```
[placement_database]
connection = mysql+pymysql://root:secret@127.0.0.1/placement?charset=utf8
```

Note: You may choose the location of the configuration file on the command line when using the `placement-manage` command.

Make the `placement-manage` command available by updating a virtualenv:

```
tox -epy36 --notest
```

Run the command to create the tables:

```
.tox/py36/bin/placement-manage db sync
```

You can confirm the tables are there with `mysqlshow placement`

Run The Service

Now we want to run the service. We need to update `placement.conf` so it will produce debugging output and use the `noauth` strategy for authentication (so we don't also have to run Keystone). Make `placement.conf` look like this (adjusting for your database settings):

```
[DEFAULT]
debug = True

[placement_database]
connection = mysql+pymysql://root:secret@127.0.0.1/placement?charset=utf8

[api]
auth_strategy = noauth2
```

We need to install the uwsgi package into the virtualenv:

```
.tox/py36/bin/pip install uwsgi
```

And then use uwsgi to run the service. Start it with:

```
.tox/py36/bin/uwsgi --http :8000 --wsgi-file .tox/py36/bin/placement-api --
↪processes 2 --threads 10
```

Note: Adjust `processes` and `threads` as required. If you do not provide these arguments the server will be a single process and thus perform poorly.

If that worked you'll see lots of debug output and `spawned uWSGI worker`. Test that things are working from another terminal with curl:

```
curl -v http://localhost:8000/
```

Get a list of resource providers with (the `x-auth-token` header is required, `openstack-api-version` is optional but makes sure we are getting the latest functionality):

```
curl -H 'x-auth-token: admin' \
     -H 'openstack-api-version: placement latest' \
     http://localhost:8000/resource_providers
```

The result ought to look something like this:

```
{"resource_providers": []}
```

If it doesn't then something went wrong with the above and there should be more information in the terminal where `uwsgi` is running.

From here you can experiment with creating resource providers and related placement features. If you change the placement code, `ctrl-c` to kill the uwsgi process and start it up again. For testing, you might enjoy placecat.

Here's all of the above as single script. As stated above this is for illustrative purposes. You should make your own:

```
#!/bin/bash

set -xe

# Change these as required
CONF_DIR=/etc/placement
DB_DRIVER=mysql+pymysql # we assume mysql throughout, feel free to change
DB_NAME=placement
DB_USER=root
DB_PASS=secret

REPO=https://opendev.org/openstack/placement

# Create a directory for configuration to live.
[[ -d $CONF_DIR ]] || (sudo mkdir $CONF_DIR && sudo chown $USER $CONF_DIR)

# Establish database. Some of this may need sudo powers. Don't be shy
# about changing the script.
mysql -u$DB_USER -p$DB_PASS -e "DROP DATABASE IF EXISTS $DB_NAME;"
mysql -u$DB_USER -p$DB_PASS -e "CREATE DATABASE $DB_NAME CHARACTER SET utf8;"
mysql -u$DB_USER -p$DB_PASS -e "GRANT ALL PRIVILEGES ON $DB_NAME.* TO '$DB_
↪USER'@'%' IDENTIFIED BY '$DB_PASS';"

# clone the right code
git clone $REPO
cd placement

# establish virtenv
tox -epy36 --notest

# write placement.conf
```

(continues on next page)

(continued from previous page)

```
cat<<EOF > $CONF_DIR/placement.conf
[DEFAULT]
debug = True

[placement_database]
connection = $DB_DRIVER://${DB_USER}:${DB_PASS}@127.0.0.1/${DB_NAME}?
↪charset=utf8

[api]
auth_strategy = noauth2
EOF

# Create database tables
.tox/py36/bin/placement-manage db sync

# install uwsgi
.tox/py36/bin/pip install uwsgi

# run uwsgi
.tox/py36/bin/uwsgi --http :8000 --wsgi-file .tox/py36/bin/placement-api --
↪processes 2 --threads 10
```

4.1.6 Testing Placement

Most of the handler code in the placement API is tested using gabbi. Some utility code is tested with unit tests found in *placement/tests/unit*. The back-end objects are tested with a combination of unit and functional tests found in `placement/tests/unit/objects` and `placement/tests/functional/db`.

When writing tests for handler code (that is, the code found in `placement/handlers`) a good rule of thumb is that if you feel like there needs to be a unit test for some of the code in the handler, that is a good sign that the piece of code should be extracted to a separate method. That method should be independent of the handler method itself (the one decorated by the `wsgify` method) and testable as a unit, without mocks if possible. If the extracted method is useful for multiple resources consider putting it in the `util` package.

As a general guide, handler code should be relatively short and where there are conditionals and branching, they should be reachable via the gabbi functional tests. This is merely a design goal, not a strict constraint.

Using Gabbi

Gabbi was developed in the telemetry project to provide a declarative way to test HTTP APIs that preserves visibility of both the request and response of the HTTP interaction. Tests are written in YAML files where each file is an ordered suite of tests. Fixtures (such as a database) are set up and torn down at the beginning and end of each file, not each test. JSON response bodies can be evaluated with JSONPath. The placement WSGI application is run via wsgi-intercept, meaning that real HTTP requests are being made over a file handle that appears to Python to be a socket.

In the placement API the YAML files (aka "gabbits") can be found in `placement/tests/functional/gabbits`. Fixture definitions are in `placement/tests/functional/fixtures/gabbits.py`. Tests are frequently grouped by handler name (e.g., `resource-provider.yaml` and `inventory.yaml`). This is not a requirement and as we increase the number of tests it makes sense to have more YAML files with fewer tests, divided up by the arc of API interaction that they test.

The gabbi tests are integrated into the functional tox target, loaded via `placement/tests/functional/test_api.py`. If you want to run just the gabbi tests one way to do so is:

```
tox -efunctional test_api
```

If you want to run just one yaml file (in this example `inventory.yaml`):

```
tox -efunctional api.inventory
```

It is also possible to run just one test from within one file. When you do this every test prior to the one you asked for will also be run. This is because the YAML represents a sequence of dependent requests. Select the test by using the name in the yaml file, replacing space with _:

```
tox -efunctional api.inventory_post_new_ipv4_address_inventory
```

Note: `tox.ini` in the placement repository is configured by a `group_regex` so that each gabbi YAML is considered a group. Thus, all tests in the file will be run in the same process when running stestr concurrently (the default).

Writing More Gabbi Tests

The docs for gabbi try to be complete and explain the syntax in some depth. Where something is missing or confusing, please log a bug.

While it is possible to test all aspects of a response (all the response headers, the status code, every attribute in a JSON structure) in one single test, doing so will likely make the test harder to read and will certainly make debugging more challenging. If there are multiple things that need to be asserted, making multiple requests is reasonable. Since database set up is only happening once per file (instead of once per test) and since there is no TCP overhead, the tests run quickly.

While fixtures can be used to establish entities that are required for tests, creating those entities via the HTTP API results in tests which are more descriptive. For example the `inventory.yaml` file creates the resource provider to which it will then add inventory. This makes it easy to explore a sequence of interactions and a variety of responses with the tests:

- create a resource provider
- confirm it has empty inventory
- add inventory to the resource provider (in a few different ways)
- confirm the resource provider now has inventory
- modify the inventory
- delete the inventory
- confirm the resource provider now has empty inventory

Nothing special is required to add a new set of tests: create a YAML file with a unique name in the same directory as the others. The other files can provide examples. Gabbi can provide a useful way of doing test driven development of a new handler: create a YAML file that describes the desired URLs and behavior and write the code to make it pass.

It's also possible to use gabbi against a running placement service, for example in devstack. See gabbi-run to get started. If you don't want to go to the trouble of using devstack, but do want a live server see *Quick Placement Development*.

Profiling

If you wish to profile requests to the placement service, to get an idea of which methods are consuming the most CPU or are being used repeatedly, it is possible to enable a ProfilerMiddleware to output per-request python profiling dumps. The environment (*Quick Placement Development* is a good place to start) in which the service is running will need to have Werkzeug added.

- If the service is already running, stop it.
- Install Werkzeug.
- Set an environment variable, `OS_WSGI_PROFILER`, to a directory where profile results will be written.
- Make sure the directory exists.
- Start the service, ensuring the environment variable is passed to it.
- Make an HTTP request that exercises the code you wish to profile.

The profiling results will be in the directory named by `OS_WSGI_PROFILER`. There are many ways to analyze the files. See Profiling WSGI Apps for an example.

Profiling with OSProfiler

To use OSProfiler with placement:

- Add a [profiler] section to the placement.conf:

```
[profiler]
connection_string = mysql+pymysql://root:admin@127.0.0.1/osprofiler?
↪charset=utf8
hmac_keys = my-secret-key
enabled = True
```

- Include the hmac_keys in your API request:

```
$ openstack resource provider list --os-profile my-secret-key
```

  The openstack client will return the trace id:

```
Trace ID: 67428cdd-bfaa-496f-b430-507165729246
```

- Extract the trace in html format:

```
$ osprofiler trace show --html 67428cdd-bfaa-496f-b430-507165729246 \
  --connection-string mysql+pymysql://root:admin@127.0.0.1/osprofiler?
↪charset=utf8
```

4.1.7 Vision Reflection

In late-2018, the OpenStack Technical Committee composed a technical vision of what OpenStack clouds should look like. This document compares the state of placement relative to that vision to provide some guidance on broad stroke ways in which placement may need to change to match the vision.

Since placement is primarily a back-end and admin-only system (at least for now), many aspects of the vision document do not apply, but it is still a useful exercise.

Note that there is also a placement *Goals* document.

The vision document is divided into three sections, which this document mirrors. This should be a living document which evolves as placement itself evolves.

The Pillars of Cloud

The sole interface to the placement service is an HTTP API, meaning that in theory, anything can talk to it, enabling the self-service and application control that define a cloud. However, at the moment the data managed by placement is considered for administrators only. This policy could be changed, but doing so would be a dramatic adjustment in the scope of who placement is for and what it does. Since placement has not yet fully satisfied its original vision to clarify and ease cloud resource allocation such a change should be considered secondary to completing the original goals.

OpenStack-specific Considerations

Placement uses microversions to help manage interoperability and bi-directional compatibility. Because placement has used microversions from the very start a great deal of the valuable functionality is only available in an opt-in fashion. In fact, it would be accurate to say that a placement service at the default microversion is incapable of being a placement service. We may wish to evaluate (and publish) if there is a minimum microversion at which placement is useful. To some extent this is already done with the way nova requires specific placement microversions, and for placement to be upgraded in advance of nova.

As yet, placement provides no dedicated mechanism for partitioning its resource providers amongst regions. Aggregates can be used for this purpose but this is potentially cumbersome in the face of multi-region use cases where a single placement service is used to manage resources in several clouds. This is an area that is already under consideration, and would bring placement closer to matching the "partitioning" aspect of the vision document.

Design Goals

Placement already maps well to several of the design goals in the vision document, adhering to fairly standard methods for scalability, reliability, customization, and flexible utilization models. It does this by being a simple web app over a database and not much more. We should strive to keep this. Details of how we plan to do so should be maintained in the *Goals* document.

CHAPTER

FIVE

SPECIFICATIONS

5.1 Placement Specifications

Significant feature developments are tracked in documents called specifications. From the Train cycle onward, those documents are kept in this section. Prior to that, Placement specifications were a part of the Nova Specs.

The following specifications represent the stages of design and development of resource providers and the Placement service. Implementation details may have changed or be partially complete at this time.

- Generic Resource Pools
- Compute Node Inventory
- Resource Provider Allocations
- Resource Provider Base Models
- Nested Resource Providers
- Custom Resource Classes
- Scheduler Filters in DB
- Scheduler claiming resources to the Placement API
- The Traits API - Manage Traits with ResourceProvider
- Request Traits During Scheduling
- filter allocation candidates by aggregate membership
- perform granular allocation candidate requests
- inventory and allocation data migration (reshaping provider trees)
- handle allocation updates in a safe way

5.1.1 Train

Implemented

Support filtering by forbidden aggregate membership

https://storyboard.openstack.org/#!/story/2005297

This blueprint proposes to support for negative filtering by the underlying resource provider's aggregate membership.

Problem description

Placement currently supports `member_of` query parameters for the `GET /resource_providers` and `GET /allocation_candidates` endpoints. This parameter is either "a string representing an aggregate uuid" or "the prefix `in:` followed by a comma-separated list of strings representing aggregate uuids".

For example:

```
&member_of=in:<agg1>,<agg2>&member_of=<agg3>
```

would translate logically to:

"Candidate resource providers should be in either agg1 or agg2, but definitely in agg3." (See alloc-candidates-member-of spec for details)

However, there is no expression for forbidden aggregates in the API. In other words, we have no way to say "don't use resource providers in this special aggregate for non-special workloads".

Use Cases

This feature is useful to save special resources for specific users.

Use Case 1

Some of the compute host are *Licensed Windows Compute Host*, meaning any VMs booted on this compute host will be considered as licensed Windows image and depending on the usage of VM, operator will charge it to the end-users. As an operator, I want to avoid booting images/volumes other than Windows OS on *Licensed Windows Compute Host*.

Use Case 2

Reservation projects like blazar would like to have its own aggregate for host reservation in order to have consumers without any reservations to be scheduled outside of that aggregate in order to save the reserved resources.

Proposed change

Adjust the handling of the `member_of` parameter so that aggregates can be expressed as forbidden. Forbidden aggregates are prefixed with a `!`.

In the following example:

```
&member_of=!<agg1>
```

would translate logically to:

"Candidate resource providers should *not* be in agg1"

This negative expression can also be used in multiple `member_of` parameters:

```
&member_of=in:<agg1>,<agg2>&member_of=<agg3>&member_of=!<agg4>
```

would translate logically to:

"Candidate resource providers must be at least one of agg1 or agg2, definitely in agg3 and definitely *not* in agg4."

Note that we don't support `!` for arguments to the `in:` prefix:

```
&member_of=in:<agg1>,<agg2>,!<agg3>
```

This would result in HTTP 400 Bad Request error.

Instead, we support `!in:` prefix:

```
&member_of=!in:<agg1>,<agg2>,<agg3>
```

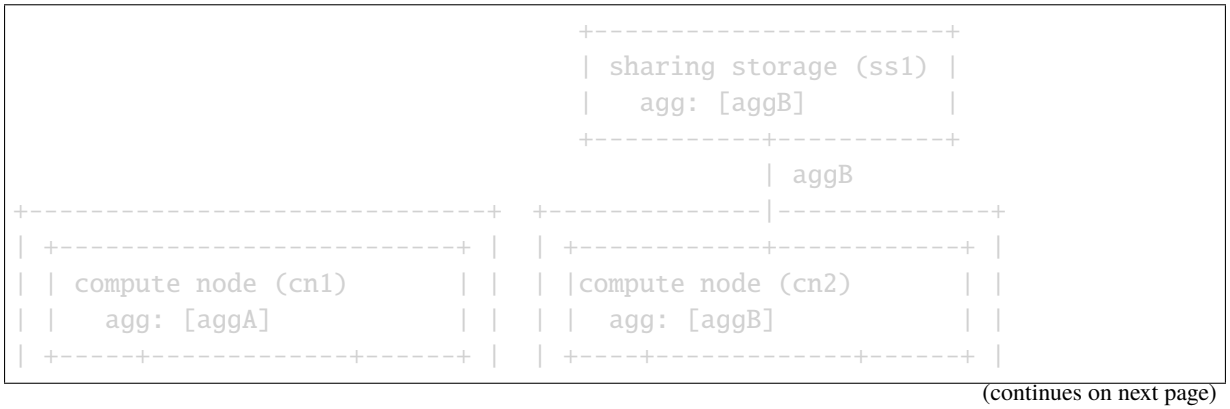
which is equivalent to:

```
member_of=!<agg1>&member_of=!<agg2>&member_of=!<agg3>
```

Nested resource providers

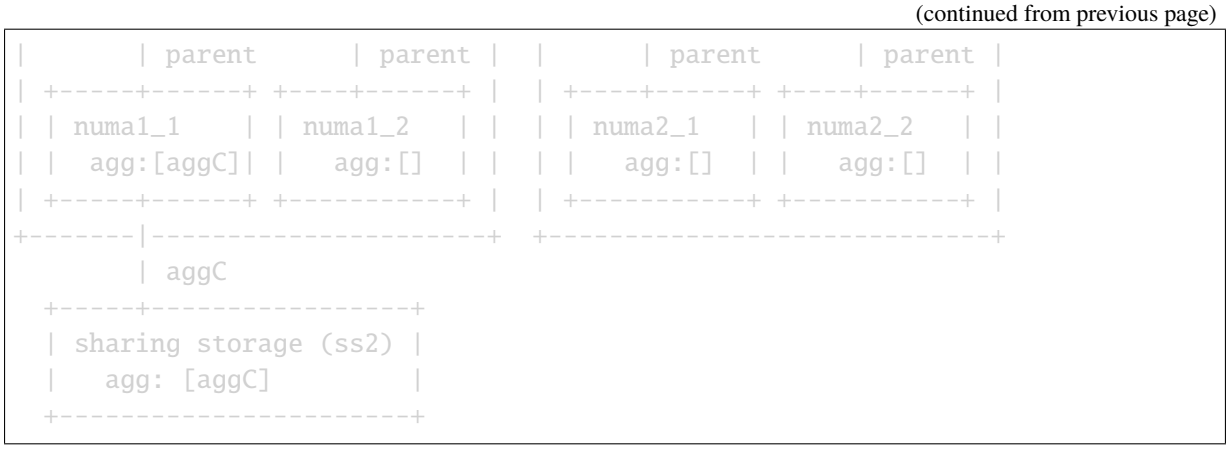
For nested resource providers, an aggregate on a root provider automatically spans the whole tree. When a root provider is in forbidden aggregates, the child providers can't be a candidate even if the child provider belongs to no (or another different) aggregate.

In the following environments, for example,



(continues on next page)

(continued from previous page)



the exclusion constraint is as follows:

- `member_of=!<aggA>` excludes "cn1", "numa1_1" and "numa1_2".
- `member_of=!<aggB>` excludes "cn2", "numa2_1", "numa2_2", and "ss1".
- `member_of=!<aggC>` excludes "numa1_1" and "ss2".

Note that this spanning doesn't happen on numbered `member_of` parameters, which is used for the granular request:

- `member_of<N>=!<aggA>` excludes "cn1"
- `member_of<N>=!<aggB>` excludes "cn2" and "ss1"
- `member_of<N>=!<aggC>` excludes "numa1_1" and "ss2".

See granular-resource-request spec for details.

Alternatives

We can use forbidden traits to exclude specific resource providers, but if we use traits, then we should put Blazar or windows license trait not only on root providers but also on every resource providers in the tree, so we don't take this way.

We can also create nova scheduler filters to do post-processing of compute hosts by looking at host aggregate relationships just as BlazarFilter does today. However, this is inefficient and we don't want to develop/use another filter for the windows license use case.

Data model impact

None.

REST API impact

A new microversion will be created which will update the validation for the `member_of` parameter on `GET /allocation_candidates` and `GET /resource_providers` to accept `!` both as a prefix on aggregate uuids and as a prefix to the `in:` prefix to express that the prefixed aggregate (or the aggregates) is to be excluded from the results.

We do not return 400 if an agg UUID is found on both the positive and negative sides of the request. For example:

```
&member_of=in:<agg1>,<agg2>&member_of=!<agg2>
```

The first member_of would return all resource_providers in either agg1 or agg2, while the second member_of would eliminate those in agg2. The result will be a 200 containing just those resource_providers in agg1. Likewise, we do not return 400 for cases like:

```
&member_of=<agg1>&member_of=!<agg1>
```

As in the previous example, we return 200 with empty results, since this is a syntactically valid request, even though a resource provider cannot be both inside and outside of agg1 at the same time.

Security impact

None.

Notifications impact

None.

Other end user impact

None.

Performance Impact

Queries to the database will see a moderate increase in complexity but existing table indexes should handle this with aplomb.

Other deployer impact

None.

Developer impact

This helps us to develop a simple reservation mechanism without having a specific nova filter, for example, via the following flow:

0. Operator who wants to enable blazar sets default forbidden and required membership key in the `nova.conf`.
    - The parameter key in the configuration file is something like `[scheduler]/placement_req_default_forbidden_member_prefix` and the value is set by the operator to `reservation:`.
    - The parameter key in the configuration file is something like `[scheduler]/placement_req_required_member_prefix` and the value would is set by the operator to `reservation:`.
1. Operator starts up the service and makes a host-pool for reservation via blazar API
    - Blazar makes an nova aggregate with `reservation:<random_id>` metadata on initialization as a blazar's free pool
    - Blazar puts hosts specified by the operator into the free pool aggregate on demand
2. User uses blazar to make a host reservation and to get the reservation id
    - Blazar picks up a host from the blazar's free pool
    - Blazar creates a new nova aggregate for that reservation and set that aggregate's metadata key to `reservation:<resv_id>` and puts the reserved host into that aggregate
3. User creates a VM with a flavor/image with `reservation:<resv_id>` meta_data/extra_specs to consume the reservation
    - Nova finds in the flavor that the extra_spec has a key which starts with what is set in `[scheduler]/placement_req_required_member_prefix`, and looks up the table for aggregates which has the specified metadata:

```
required_prefix = CONF.scheduler.placement_req_required_member_prefix
# required_prefix = 'reservation:'
required_meta_data = get_flavor_extra_spec_starts_with(required_
↪prefix)
# required_meta_data = 'reservation:<resv_id>'
required_aggs = aggs_whose_metadata_is(required_meta_data)
# required_aggs = [<An aggregate for the reservation>]
```

    - Nova finds out that the default forbidden aggregate metadata prefix, which is set in `[scheduler]/placement_req_default_forbidden_member_prefix`, is explicitly via the flavor, so skip:

```
default_forbidden_prefix = CONF.scheduler.placement_req_default_
↪forbidden_member_prefix
# default_forbidden_prefix = ['reservation:']
forbidden_aggs = set()
if not get_flavor_extra_spec_starts_with(default_forbidden_prefix):
```

(continues on next page)

(continued from previous page)

```
    # this is skipped because 'reservation:' is in the flavor in this␣
↪case
    forbidden_aggs = aggs_whose_metadata_starts_with(default_
↪forbidden_prefix)
```

- Nova calls placement with required and forbidden aggregates:

```
# We don't have forbidden aggregates in this case
?member_of=<required_aggs>
```

4. User creates a VM with a flavor/image with no reservation, that is, without `reservation:<resv_id>` meta_data/extra_specs.

   - Nova finds in the flavor that the extra_spec has no key which starts with what is set in `[scheduler]/placement_req_required_member_prefix`, so no required aggregate is obtained:

```
required_prefix = CONF.scheduler.placement_req_required_member_prefix
# required_prefix = 'reservation:'
required_meta_data = get_flavor_extra_spec_starts_with(required_
↪prefix)
# required_meta_data = ''
required_aggs = aggs_whose_metadata_is(required_meta_data)
# required_aggs = set()
```

   - Nova looks up the table for default forbidden aggregates whose metadata starts with what is set in `[scheduler]/placement_req_default_forbidden_member_prefix`:

```
default_forbidden_prefix = CONF.scheduler.placement_req_default_
↪forbidden_member_prefix
# default_forbidden_prefix = ['reservation:']
forbidden_aggs = set()
if not get_flavor_extra_spec_starts_with(default_forbidden_prefix):
    # This is not skipped now
    forbidden_aggs = aggs_whose_metadata_starts_with(default_
↪forbidden_prefix)
# forbidden_aggs = <blazar's free pool aggregates and the other␣
↪reservation aggs>
```

   - Nova calls placement with required and forbidden aggregates:

```
# We don't have required aggregates in this case
?member_of=!in:<forbidden_aggs>
```

Note that the change in the nova configuration file and change in the request filter is an example and out of the scope of this spec. An alternative for this is to let placement be aware of the default forbidden traits/aggregates (See the Bi-directional enforcement of traits spec). But we agreed that it is not placement but nova which is responsible for what traits/aggregate is forbidden/required for the instance.

Upgrade impact

None.

Implementation

Assignee(s)

Primary assignee: Tetsuro Nakamura (nakamura.tetsuro@lab.ntt.co.jp)

Work Items

- Update the `ResourceProviderList.get_all_by_filters` and `AllocationCandidates.get_by_requests` methods to change the database queries to filter on "not this aggregate".
- Update the placement API handlers for `GET /resource_providers` and `GET /allocation_candidates` in a new microversion to pass the negative aggregates to the methods changed in the steps above, including input validation adjustments.
- Add functional tests of the modified database queries.
- Add gabbi tests that express the new queries, both successful queries and those that should cause a 400 response.
- Release note for the API change.
- Update the microversion documents to indicate the new version.
- Update placement-api-ref to show the new query handling.

Dependencies

None.

Testing

Normal functional and unit testing.

Documentation Impact

Document the REST API microversion in the appropriate reference docs.

References

- alloc-candidates-member-of feature
- granular-resource-request feature

History

Table 1: Revisions

Release Name	Description
Stein	Approved but not implemented
Train	Reproposed

Getting On The Nested Magic Train 1

https://storyboard.openstack.org/#!/story/2005575

This spec describes a cluster of Placement API work to support several interrelated use cases for Train around:

- Modeling complex trees such as NUMA layouts, multiple devices, networks.
- Requesting affinity[1] between/among the various providers/allocations in allocation candidates against such layouts.
- Describing granular groups more richly to facilitate the above.
- Requesting candidates based on traits that are not necessarily associated with resources.

An additional spec, for a feature known as can_split has been separated out to its own spec to ensure that any delay in it does not impact these features, which are less controversial.

Principles

In developing this design, some fundamental concepts have come to light. These are not really changes from the existing architecture, but understanding them becomes more important in light of the changes introduced herein.

Resource versus Provider Traits

The database model associates traits with resource providers, not with inventories of resource classes. However, conceptually there are two different categories of traits to consider.

Resource Traits are tied to specific resources. For example, `HW_CPU_X86_AVX2` describes a characteristic of `VCPU` (or `PCPU`) resources.

Provider Traits are characteristics of a provider, regardless of the resources it provides. For example, `COMPUTE_VOLUME_MULTI_ATTACH` is a capability of a compute host, not of any specific resource inventory. `HW_NUMA_ROOT` describes NUMA affinity among *all* the resources in the inventories of that

[1] The kind of affinity we're talking about is best understood by referring to the use case for the *same_subtree* feature below.

provider *and* all its descendants. `CUSTOM_PHYSNET_PUBLIC` indicates connectivity to the `public` network, regardless of whether the associated resources are `VF`, `PF`, `VNIC`, etc.; and regardless of whether those resources reside on the provider marked with the trait or on its descendants.

This distinction becomes important when deciding how to model. **Resource traits** need to "follow" their resource class. For example, `HW_CPU_X86_AVX2` should be on the provider of `VCPU` (or `PCPU`) resource, whether that's the root or a NUMA child. On the other hand, **provider traits** must stick to their provider, regardless of where resources inventories are placed. For example, `COMPUTE_VOLUME_MULTI_ATTACH` should always be on the root provider, as the root provider conceptually represents "the compute host".

Alternative: "Traits Flow Down": There have been discussions around a provider implicitly inheriting the traits of its parent (and therefore all its ancestors). This would (mostly) allow us not to think about the distinction between "resource" and "provider" traits. We ultimately decided against this by a hair, mainly because of this:

> It makes no sense to say my PGPU is capable of MULTI_ATTACH
>
> In addition, IIUC, there are SmartNICs [1] that have CPUs on cards. If someone will want to report/model those CPUs in placement, they will be scared that CPU traits on compute side flow down to those CPUs on NIC despite they are totally different CPUs.
>
> [1] https://www.netronome.com/products/smartnic/overview/

...and because we were able to come up with other satisfactory solutions to our use cases.

Group-Specific versus Request-Wide Query Parameters

granular resource requests introduced a divide between `GET /allocation_candidates` query parameters which apply to a particular request group

- resources[$S]
- required[$S]
- member_of[$S]
- in_tree[$S]

...and those which apply to the request as a whole

- limit
- group_policy

This has been fairly obvious thus far; but this spec introduces concepts (such as *root_required* and *same_subtree*) that make it important to keep this distinction in mind. Moving forward, we should consider whether new features and syntax additions make more sense to be group-specific or request-wide.

Proposed change

All changes are to the `GET /allocation_candidates` operation via new microversions, one per feature described below.

arbitrary group suffixes

Use case: Client code managing request groups for different kinds of resources - which will often come from different providers - may reside in different places in the codebase. For example, the management of compute resources vs. networks vs. accelerators. However, there still needs to be a way for the consuming code to express relationships (such as affinity) among these request groups. For this purpose, API consumers wish to be able to use conventions for request group identifiers. It would also be nice for development and debugging purposes if these designations had some element of human readability.

(Merged) code is here: https://review.opendev.org/#/c/657419/

Granular groups are currently restricted to using integer suffixes. We will change this so they can be case-sensitive strings up to 64 characters long comprising alphanumeric (either case), underscore, and hyphen.

- 64c so we can fit a stringified UUID (with hyphens) as well as some kind of handy type designation. Like `resources_PORT_$UUID`. https://review.opendev.org/#/c/657419/4/placement/schemas/allocation_candidate.py@19
- We want to allow uppercase so consumers can make nice visual distinctions like `resources_PORT...`; we want to allow lowercase because openstack consumers tend to use lowercase UUIDs and this makes them not have to convert them. Placement will use the string in the form it is given and transform it neither on input nor output. If the form does not match constraints a `400` response will be returned. https://review.opendev.org/#/c/657419/4/placement/schemas/allocation_candidate.py@19
- **Alternative** Uppercase only so we don't have to worry about case sensitivity or confusing differentiation from the prefixes (which are lowercase). **Rejected** because we prefer allowing lowercase UUIDs, and are willing to give the consumer the rope. https://review.opendev.org/#/c/657419/1/placement/lib.py@31
- Hyphens so we can use UUIDs without too much scrubbing.

For purposes of documentation (and this spec), we'll rename the "unnumbered" group to "unspecified" or "unsuffixed", and anywhere we reference "numbered" groups we can call them "suffixed" or "granular" (I think this label is already used in some places).

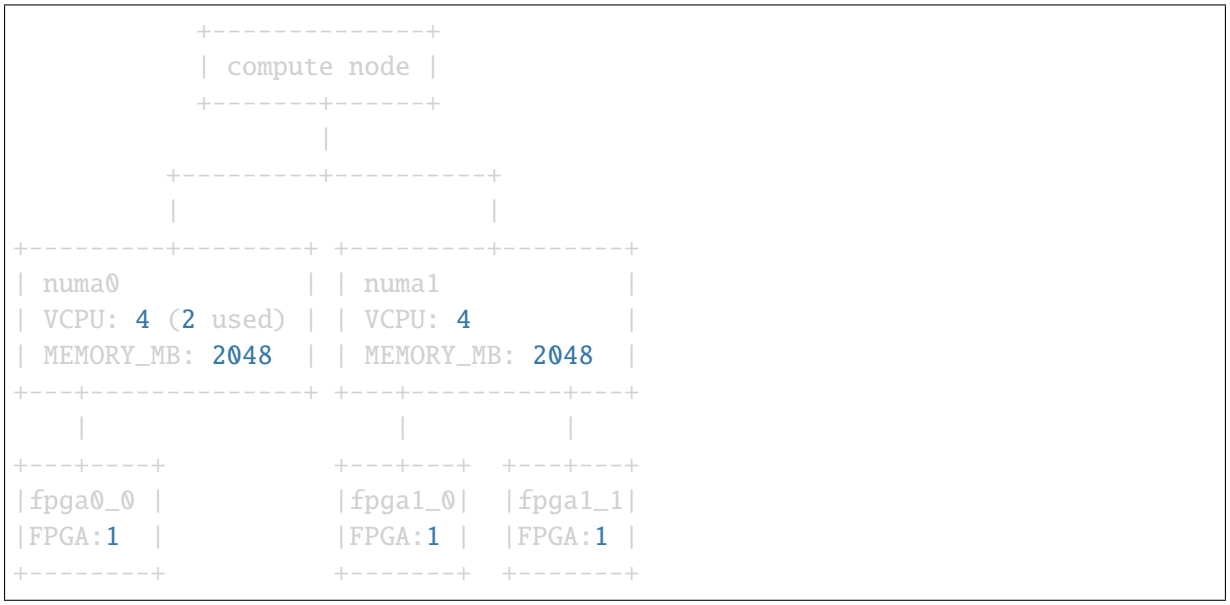
same_subtree

Use case: I want to express affinity between/among allocations in separate request groups. For example, that a `VGPU` come from a GPU affined to the NUMA node that provides my `VCPU` and `MEMORY_MB`; or that multiple network `VF`s come from the same NIC.

A new `same_subtree` query parameter will be accepted. The value is a comma-separated list of request group suffix strings `$S`. Each must exactly match a suffix on a granular group somewhere else in the request. Importantly, the identified request groups need not have a `resources$S` (see *resourceless request groups*).

We define "same subtree" as "all of the resource providers satisfying the request group must be rooted at one of the resource providers satisfying the request group". Or put another way: "one of the resource providers satisfying the request group must be the direct ancestor of all the other resource providers satisfying the request group".

For example, given a model like:



to request "two VCPUs, 512MB of memory, and one FPGA from the same NUMA node," my request could include:

```
?resources_COMPUTE=VCPU:2,MEMORY_MB:512
&resources_ACCEL=FPGA:1
# NOTE: The suffixes include the leading underscore!
&same_subtree=_COMPUTE,_ACCEL
```

This will produce candidates including:

```
- numa0: {VCPU:2, MEMORY_MB:512}, fpga0_0: {FPGA:1}
- numa1: {VCPU:2, MEMORY_MB:512}, fpga1_0: {FPGA:1}
- numa1: {VCPU:2, MEMORY_MB:512}, fpga1_1: {FPGA:1}
```

but *not*:

```
- numa0: {VCPU:2, MEMORY_MB:512}, fpga1_0: {FPGA:1}
- numa0: {VCPU:2, MEMORY_MB:512}, fpga1_1: {FPGA:1}
- numa1: {VCPU:2, MEMORY_MB:512}, fpga0_0: {FPGA:1}
```

The `same_subtree` query parameter is *request-wide*, but may be repeated. Each grouping is treated independently.

Anti-affinity

There were discussions about supporting `!` syntax in `same_subtree` to express anti-affinity (e.g. `same_subtree=$X,!$Y` meaning "resources from group `$Y` shall *not* come from the same subtree as resources from group `$X`"). This shall be deferred to a future release.

resourceless request groups

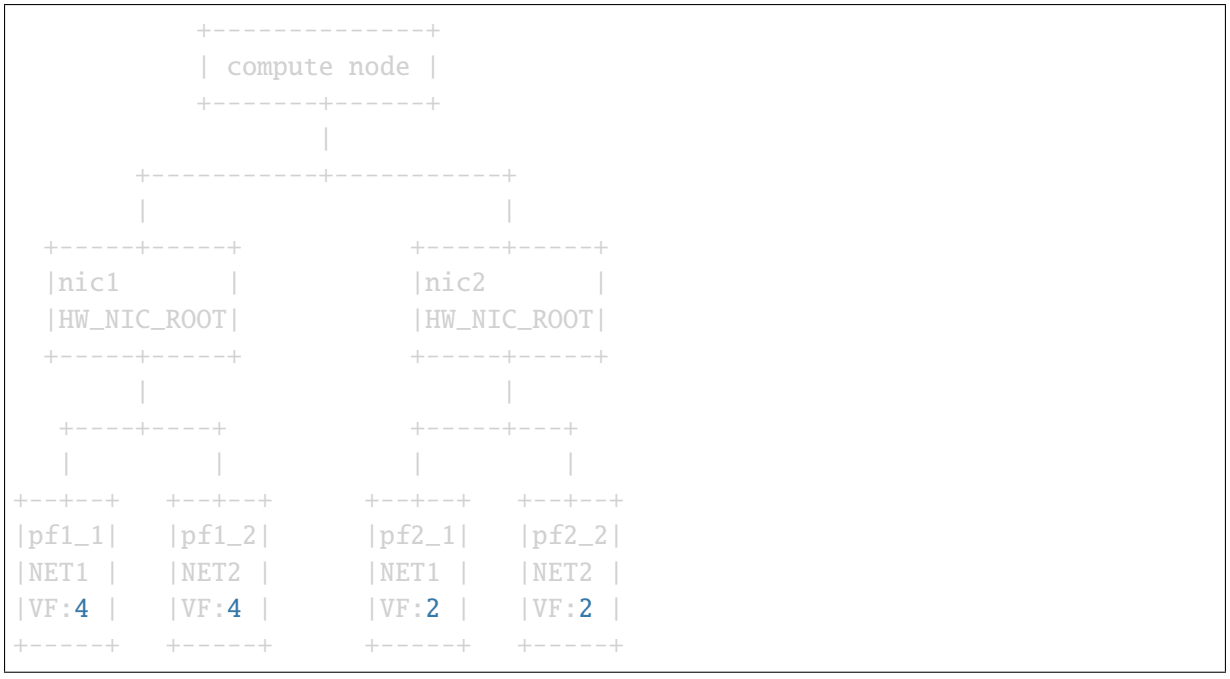
Use case: When making use of *same_subtree*, I want to be able to identify a provider as a placeholder in the subtree structure even if I don't need any resources from that provider.

It is currently a requirement that a `resources$S` exist for all `$S` in a request. This restriction shall be removed such that a request group may exist e.g. with only `required$S` or `member_of$S`.

There must be at least one `resources` or `resources$S` somewhere in the request, otherwise there will be no inventory to allocate and thus no allocation candidates. If neither is present a `400` response will be returned.

Furthermore, resourceless request groups must be used with *same_subtree*. That is, the suffix for each resourceless request group must feature in a `same_subtree` somewhere in the request. Otherwise a `400` response will be returned. (The reasoning for this *restriction* is explained below.)

For example, given a model like:



a request such as the following, meaning, "Two VFs from the same NIC, one on each of network NET1 and NET2," is legal:

```
?resources_VIF_NET1=VF:1
&required_VIF_NET1=NET1
&resources_VIF_NET2=VF:1
&required_VIF_NET2=NET2
# NOTE: there is no resources_NIC_AFFINITY
```

(continues on next page)

(continued from previous page)

```
&required_NIC_AFFINITY=HW_NIC_ROOT
&same_subtree=_VIF_NET1,_VIF_NET2,_NIC_AFFINITY
```

The returned candidates will include:

```
- pf1_1: {VF:1}, pf1_2: {VF:1}
- pf2_1: {VF:1}, pf2_2: {VF:1}
```

but *not*:

```
- pf1_1: {VF:1}, pf2_2: {VF:1}
- pf2_1: {VF:1}, pf1_2: {VF:1}
```

Why enforce resourceless + same_subtree?

Taken by itself (without *same_subtree*), a resourceless request group intuitively means, "There must exist in the solution space a resource provider that satisfies these constraints." But what does "solution space" mean? Clearly it's not the same as *solution path*, or we wouldn't be able to use it to add resourceless providers to that solution path. So it must encompass at least the entire non-sharing tree around the solution path. Does it also encompass sharing providers associated via aggregate? What would that mean?

Since we have not identified any real use cases for resourceless *without* *same_subtree* (other than *root_member_of* -- see below) making this an error allows us to not have to deal with these questions.

root_required

Use case: I want to limit allocation candidates to trees *whose root provider* has (or does not have) certain traits. For example, I want to limit candidates to only multi-attach-capable hosts; or preserve my Windows-licensed hosts for special use.

A new `root_required` query parameter will be accepted. The value syntax is identical to that of `required[$S]`: that is, it accepts a comma-delimited list of trait names, each optionally prefixed with `!` to indicate "forbidden" rather than "required".

This is a *request-wide* query parameter designed for *provider traits* specifically on the root provider of the non-sharing tree involved in the allocation candidate. That is, regardless of any group-specific constraints, and regardless of whether the root actually provides resource to the request, results will be filtered such that the root of the non-sharing tree conforms to the constraints specified in `root_required`.

`root_required` may not be repeated.

The fact that this feature is (somewhat awkwardly) restricted to "...trees whose root provider ..." deserves some explanation. This is to fill a gap in use cases that cannot be adequately covered by other query parameters.

- To land on a tree (host) with a given trait *anywhere* in its hierarchy, *resourceless request groups* without *same_subtree* could be used. However, there is no way to express the "forbidden" side of this in a way that makes sense:
  - A resourceless `required$S=!FOO` would simply ensure that a provider *anywhere in the tree* does not have `FOO` - which would end up not being restrictive as intended in most cases.

  - We could define "resourceless forbidden" to mean "nowhere in the tree", but this would be inconsistent and hard to explain.

- To ensure that the desired trait is present (or absent) in the *result set*, it would be necessary to attach the trait to a group whose resource constraints will be satisfied by the provider possessing (or lacking) that trait.

  - This requires the API consumer to understand too much about how the provider trees are modeled; and

  - It doesn't work in heterogeneous environments where such *provider traits* may or may not stick with providers of a specific resource class.

  This could possibly be mitigated by careful use of *same_subtree*, but that again requires deep understanding of the tree model, and also confuses the meaning of *same_subtree* and *resource versus provider traits*.

- The *traits flow down* concept described earlier could help here; but that would still entail attaching *provider traits* to a particular request group. Which one? Because the trait isn't associated with a specific resource, it would be arbitrary and thus difficult to explain and justify.

Alternative: "Solution Path": A more general solution was discussed whereby we would define a "solution path" as: **The set of resource providers which satisfy all the request groups *plus* all the ancestors of those providers, up to the root.** This would allow us to introduce a *request-wide* query parameter such as `solution_path_required`. The idea would be the same as `root_required`, but the specified trait constraints would be applied to all providers in the "solution path" (required traits must be present *somewhere* in the solution path; forbidden traits must not be present *anywhere* in the solution path).

This alternative was rejected because:

- Describing the "solution path" concept to API consumers would be hard.
- We decided the only real use cases where the trait constraints needed to be applied to providers *other than the root* could be satisfied (and more naturally) in other ways.

This section was the result of long discussions in IRC and on the review for this spec

root_member_of

Note: When this spec was initially written it was not clear whether there was immediate need to implement this feature. This turned out to be the case. The feature was not implemented in the Train cycle. It will be revisted in the future if needed.

Use case: I want to limit allocation candidates to trees *whose root provider* is (or is not) a member of a certain aggregate. For example, I want to limit candidates to only hosts in (or not in) a specific availability zone.

Note: We "need" this because of the *restriction* that resourceless request groups must be used with *same_subtree*. Without that restriction, a resourceless `member_of` would match a provider anywhere in the tree, including the root.

`root_member_of` is conceptually identical to *root_required*, but for aggregates. Like `member_of[$S]`, `root_member_of` supports `in:`, and can be repeated (in contrast to `[root_]required[$S]`).

Default group_policy to none

A single `isolate` setting that applies to the whole request has consistently been shown to be inadequate/confusing/frustrating for all but the simplest anti-affinity use cases. We're not going to get rid of `group_policy`, but we're going to make it no longer required, defaulting to `none`. This will allow us to get rid of at least one hack in nova and provide a clearer user experience, while still allowing us to satisfy simple NUMA use cases. In the future a *granular isolation* syntax should make it possible to satisfy more complex scenarios.

(Future) Granular Isolation

Note: This is currently out of scope, but we wanted to get it written down.

The features elsewhere in this spec allow us to specify affinity pretty richly. But anti-affinity (within a provider tree - not between providers) is still all (`group_policy=isolate`) or nothing (`group_policy=none`). We would like to be able to express anti-affinity between/among subsets of the suffixed groups in the request.

We propose a new *request-wide* query parameter key `isolate`. The value is a comma-separated list of request group suffix strings `$S`. Each must exactly match a suffix on a granular group somewhere else in the request. This works on *resourceless request groups* as well as those with resources. It is mutually exclusive with the `group_policy` query parameter: 400 if both are specified.

The effect is the resource providers satisfying each group `$S` must satisfy *only* their respective group `$S`.

At one point I thought it made sense for `isolate` to be repeatable. But now I can't convince myself that `isolate={set1}&isolate={set2}` can ever produce an effect different from `isolate={set1|set2}`. Perhaps it's because different `isolate`s could be coming from different parts of the calling code?

Another alternative would be to isolate the groups from *each other* but not from *other groups*, in which case repeating `isolate` could be meaningful. But confusing. Thought will be needed.

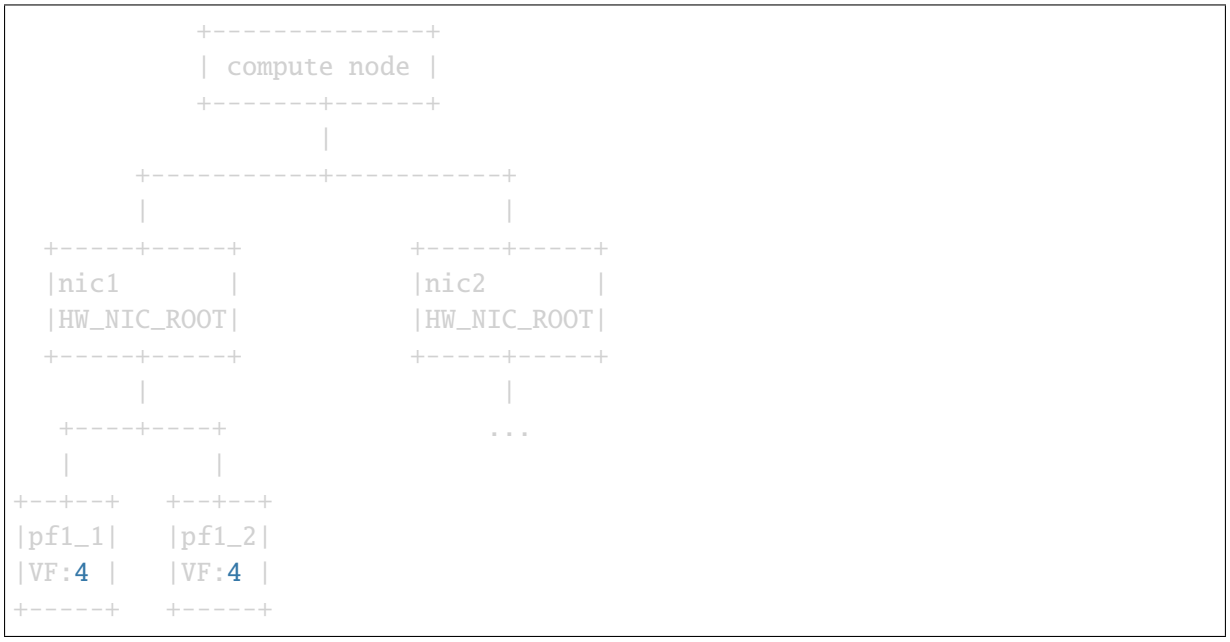
Interactions

Some discussion on these can be found in the neighborhood of http://eavesdrop.openstack.org/irclogs/%23openstack-placement/%23openstack-placement.2019-05-10.log.html#t2019-05-10T22:02:43

group_policy + same_subtree

`group_policy=isolate` forces the request groups identified in `same_subtree` to be satisfied by different providers, whereas `group_policy=none` would also allow `same_subtree` to degenerate to "same provider".

For example, given the following model:



a request for "Two VFs from different PFs on the same NIC":

```
?resources_VIF1=VF:1
&resources_VIF2=VF:1
&required_NIC_AFFINITY=HW_NIC_ROOT
&same_subtree=_VIF1,_VIF2,_NIC_AFFINITY
&group_policy=isolate
```

will return only one candidate:

```
- pf1_1: {VF:1}, pf1_2: {VF:1}
```

whereas the same request with `group_policy=none`, meaning "Two VFs from the same NIC":

```
?resources_VIF1=VF:1
&resources_VIF2=VF:1
&required_NIC_AFFINITY=HW_NIC_ROOT
&same_subtree=_VIF1,_VIF2,_NIC_AFFINITY
&group_policy=none
```

will return two additional candidates where both VFs are satisfied by the same provider:

```
- pf1_1: {VF:1}, pf1_2: {VF:1}
- pf1_1: {VF:2}
- pf1_2: {VF:2}
```

group_policy + resourceless request groups

Resourceless request groups are treated the same as any other for the purposes of `group_policy`:

- If your resourceless request group is suffixed, `group_policy=isolate` means the provider satisfying the resourceless request group will not be able to satisfy any other suffixed group.
- If your resourceless request group is unsuffixed, it can be satisfied by *any* provider in the tree, since the unsuffixed group isn't isolated (even with `group_policy=isolate`). This is important because there are cases where we want to require certain traits (usually *provider traits*), and don't want to figure out which other request group might be requesting resources from the same provider.

same_subtree + resourceless request groups

These *must* be used together -- see *Why enforce resourceless + same_subtree?*

Impacts

Data model impact

There should be no changes to database table definitions, but the implementation will almost certainly involve adding/changing database queries.

There will also likely be changes to python-side objects representing meta-objects used to manage information between the database and the REST layer. However, the data models for the JSON payloads in the REST layer itself will be unaffected.

Performance Impact

The work for `same_subtree` will probably (at least initially) be done on the python side as additional filtering under `_merge_candidates`. This could have some performance impact especially on large data sets. Again, we should optimize requests without `same_subtree`, where `same_subtree` refers to only one group, where no nested providers exist in the database, etc.

Resourceless request groups may add a small additional burden to database queries, but it should be negligible. It should be relatively rare in the wild for a resourceless request group to be satisfied by a provider that actually provides no resource to the request, though there are cases where a resourceless request group would be useful even though the provider *does* provide resources to the request.

Documentation Impact

The new query parameters will be documented in the API reference.

Microversion paperwork will be done.

Modeling with Provider Trees will be updated (and/or split off of).

Security impact

None

Other end user impact

None

Other deployer impact

None

Developer impact

None

Upgrade impact

None

Implementation

Assignee(s)

- cdent
- tetsuro
- efried
- others

Dependencies

None

Testing

Code for a gabbi fixture with some complex and interesting characteristics is merged here: https://review.opendev.org/#/c/657463/

Lots of functional testing, primarily via gabbi, will be included.

It wouldn't be insane to write some PoC consuming code on the nova side to validate assumptions and use cases.

References

...are inline

History

Table 2: Revisions

Release Name	Description
Train	Introduced

Provide resource provider - request group mapping in allocation candidates

https://blueprints.launchpad.net/nova/+spec/placement-resource-provider-request-group-mapping-in-allocation-candida

To support QoS minimum bandwidth policy during server scheduling Neutron needs to know which resource provider provides the bandwidth resource for each port in the server create request. Similar needs arise in case of handling VGPUs and accelerator devices.

Problem description

Placement supports granular request groups in the `GET allocation_candidates` query but the returned allocation candidates do not contain explicit information about which granular request group is fulfilled by which RP in the candidate. For example the resource request of a Neutron port is mapped to a granular request group by Nova towards Placement during scheduling. After scheduling Neutron needs the information about which port got allocation from which RP to set up the proper port binding towards those network device RPs. Similar examples can be created with VGPU and accelerator devices.

Doing this mapping in Nova is possible (see the current implementation) but scales pretty badly even for small amount of ports in a single server create request. See the *Non-scalable Nova based solution* section with detailed examples and analysis.

On the other hand when Placement builds an allocation candidate it does that by building allocations for each granular request group. Therefore Placement could include the necessary mapping information in the response with significantly less effort.

So doing the mapping in Nova also duplicates logic that is already implemented in Placement.

Use Cases

The use case of the bandwidth resource provider spec applies here because to fulfill that use case in a scalable way we need to consider the change proposed in this spec. Similarly handling VGPUs and accelerator devices requires this mapping information as well.

Proposed change

Extend the response of the `GET /allocation_candidates` API with an extra field `mapping` for each candidate. This field contains a mapping between resource request group names and RP UUIDs for each candidate to express which RP provides the resource for which request groups.

Alternatives

For API alternatives about the proposed REST API change see the REST API section.

Non-scalable Nova based solution

Given a single compute with the following inventories:

```
Compute RP (name=compute1, uuid=compute_uuid)
 +    CPU = 1
 |    MEMORY = 1024
 |    DISK = 10
 |
 +--+Network agent RP (for SRIOV agent),
        +    uuid=sriov_agent_uuid
        |
        |
        +--+Physical network interface RP
        |    uuid = uuid5(compute1:eth0)
        |    resources:
        |        NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=2000
        |        NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=2000
        |    traits:
        |        CUSTOM_PHYSNET_1
        |        CUSTOM_VNIC_TYPE_DIRECT
        |
        +--+Physical network interface RP
             uuid = uuid5(compute1:eth1)
             resources:
                 NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=2000
                 NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=2000
             traits:
                 CUSTOM_PHYSNET_1
                 CUSTOM_VNIC_TYPE_DIRECT
```

Example 1 - boot with a single port having bandwidth request

Neutron port:

```
{
    'id': 'da941911-a70d-4aac-8be0-c3b263e6fd4f',
    'resource_request': {
        "resources": {
            "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND": 1000,
            "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND": 1000},
        "required": ["CUSTOM_PHYSNET_1",
                     "CUSTOM_VNIC_TYPE_DIRECT"]
    }
}
```

Placement request during scheduling:

```
GET /placement/allocation_candidates?
    limit=1000&
    resources=DISK_GB=1,MEMORY_MB=512,VCPU=1&
    required1=CUSTOM_PHYSNET_1,CUSTOM_VNIC_TYPE_DIRECT&
    resources1=NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=1000,
               NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=1000
```

Placement response:

```
{
   "allocation_requests":[
      {
         "allocations":{
            uuid5(compute1:eth0):{
              "resources":{
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000
               }
            },
            compute_uuid:{
               "resources":{
                  "MEMORY_MB":512,
                  "DISK_GB":1,
                  "VCPU":1
               }
            }
         }
      },
      // ... another similar allocations with uuid5(compute1:eth1)
   ],
   "provider_summaries":{
       // ...
   }
}
```

Filter scheduler selects the first candidate that points to uuid5(compute1:eth0)

The nova-compute needs to pass RP UUID which provides resource for each port to Neutron in the port binding. To be able to do that nova (in the current implementation the nova-conductor) needs to find the RP in the selected allocation candidate which provides the resources the Neutron port is requested. The current implementation does this by checking which RP provides the matching resource classes and resource amounts.

During port binding nova updates the port with that network device RP:

```
{
  "id":"da941911-a70d-4aac-8be0-c3b263e6fd4f",
  "resource_request":{
      "resources":{
         "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000,
         "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000
      },
      "required":[
         "CUSTOM_PHYSNET_1",
         "CUSTOM_VNIC_TYPE_DIRECT"
      ]
   },
   "binding:host_id":"compute1",
   "binding:profile":{
      "allocation": uuid5(compute1:eth0)
   },
}
```

This scenario is easy as only one port is requesting bandwidth resources so there will be only one RP in the each allocation candidate that provides such resources.

Example 2 - boot with two ports having bandwidth request

Neutron port1:

```
{
    'id': 'da941911-a70d-4aac-8be0-c3b263e6fd4f',
    'resource_request': {
        "resources": {
            "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND": 1000,
            "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND": 1000},
        "required": ["CUSTOM_PHYSNET_1",
                     "CUSTOM_VNIC_TYPE_DIRECT"]
    }
}
```

Neutron port2:

```
{
    'id': '2f2613ce-95a9-490a-b3c4-5f1c28c1f886',
    'resource_request': {
```

(continues on next page)

(continued from previous page)

```
        "resources": {
            "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND": 1000,
            "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND": 2000},
        "required": ["CUSTOM_PHYSNET_1",
                     "CUSTOM_VNIC_TYPE_DIRECT"]
    }
}
```

Placement request during scheduling:

```
GET /placement/allocation_candidates?
    group_policy=isolate&
    limit=1000&
    resources=DISK_GB=1,MEMORY_MB=512,VCPU=1&
    required1=CUSTOM_PHYSNET_1,CUSTOM_VNIC_TYPE_DIRECT&
    resources1=NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=1000,
               NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=1000&
    required2=CUSTOM_PHYSNET_1,CUSTOM_VNIC_TYPE_DIRECT&
    resources2=NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=1000,
               NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=2000
```

In the above request the granular request group1 is generated from port1 and granular request group2 is generated from port2.

Placement response:

```
{
   "allocation_requests":[
      {
         "allocations":{
            uuid5(compute1:eth0):{
              "resources":{
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000
               }
            },
            uuid5(compute1:eth1):{
              "resources":{
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":2000
               }
            },
            compute_uuid:{
               "resources":{
                  "MEMORY_MB":512,
                  "DISK_GB":1,
                  "VCPU":1
               }
            }
         }
```

(continues on next page)

(continued from previous page)

```
      },
      // ... another similar allocation_request where the allocated
      // amounts are reversed between uuid5(compute1:eth0) and
      // uuid5(compute1:eth1)
   ],
   "provider_summaries":{
       // ...
   }
}
```

Filter scheduler selects the first candidate.

Nova needs to find the RP in the selected allocation candidate which provides the resources for each Neutron port request.

For the selected allocation candidate there are two possible port - RP mappings but only one valid mapping if we consider the bandwidth amounts:

- port1 - uuid5(compute1:eth0)
- port2 - uuid5(compute1:eth1)

When Nova tries to map the first port, port1, then both uuid5(compute1:eth0) and uuid5(compute1:eth1) still has enough resources in the allocation request to match with the request of port1. So at that point Nova can map port1 to uuid5(compute1:eth1). However this means that Nova will not find any viable mapping later for port2 and therefore Nova has to go back an retry to create the mapping with port1 mapped to the other alternative. This means that Nova needs to implement a full backtracking algorithm to find the proper mapping.

Scaling considerations

With 4 RPs and 4 ports, in worst case, we have 4! (24) possible mappings and each mappings needs 4 steps to be generated (assuming that in the worst case the mapping of the 4th port is the one that fails). So this backtrack makes 96 steps. So I think this code will scale pretty badly.

Note that our example uses the group_policy=isolate query param so the RPs in the allocation candidate cannot overlap. If we set group_policy=none and therefore allow RP overlapping then the necessary calculation step could grow even more.

Note that even if having more than 4 ports for an server considered unrealistic, additional granular request groups can appear in the allocation candidate request from other sources than Neutron, e.g. from flavor extra_spec due to VGPUs or from Cyborg due to accelerators.

Data model impact

None

REST API impact

Extend the response of the `GET /allocation_candidates` API with an extra field `mappings` for each candidate in a new microversion. This field contains a mapping between resource request group names and RP UUIDs for each candidate to express which RP provides the resource for which request groups.

For the request:

```
GET /placement/allocation_candidates?
    resources=DISK_GB=1,MEMORY_MB=512,VCPU=1&
    required1=CUSTOM_PHYSNET_1,CUSTOM_VNIC_TYPE_DIRECT&
    resources1=NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=1000,
               NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=1000&
    required2=CUSTOM_PHYSNET_1,CUSTOM_VNIC_TYPE_DIRECT&
    resources2=NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND=1000,
               NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND=2000
```

Placement would return the response:

```
{
   "allocation_requests":[
      {
         "allocations":{
            uuid5(compute1:eth0):{
              "resources":{
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000
               },
            },
            uuid5(compute1:eth1):{
              "resources":{
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":2000
               },
            },
            compute_uuid:{
               "resources":{
                  "MEMORY_MB":512,
                  "DISK_GB":1,
                  "VCPU":1
                },
            }
         },
         "mappings": {
             "1": [uuid5(compute1:eth0)],
             "2": [uuid5(compute1:eth1)],
```

(continues on next page)

(continued from previous page)

```
            "": [compute_uuid],
        },
    },
    {
        "allocations":{
            uuid5(compute1:eth1):{
              "resources":{
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000
               },
            },
            uuid5(compute1:eth0):{
              "resources":{
                  "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
                  "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":2000
               },
            },
            compute_uuid:{
               "resources":{
                  "MEMORY_MB":512,
                  "DISK_GB":1,
                  "VCPU":1
                },
            }
        },
        "mappings": {
            "1": [uuid5(compute1:eth1)],
            "2": [uuid5(compute1:eth0)],
            "": [compute_uuid],
        },
    },
  ],
  "provider_summaries":{
      // unchanged
  }
}
```

The numbered groups are always satisfied by a single RP so the length of the mapping value will be always 1. However the unnumbered group might be satisfied by more than one RPs so the length of the mapping value there can be bigger than 1.

This new field will be added to the schema for `POST /allocations`, `PUT /allocations/{consumer_uuid}`, and `POST /reshaper` so the client does not need to strip it from the candidate before posting that back to Placement to make the allocation. The contents of the field will be ignored by these operations.

Alternatively the mapping can be added as a separate top level key to the response.

Response:

```
{
```

(continues on next page)

(continued from previous page)

```
"allocation_requests":[
   {
      "allocations":{
         uuid5(compute1:eth0):{
           "resources":{
               "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
               "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000
            },
         },
         uuid5(compute1:eth1):{
           "resources":{
               "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
               "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":2000
            },
         },
         compute_uuid:{
            "resources":{
               "MEMORY_MB":512,
               "DISK_GB":1,
               "VCPU":1
             },
         }
      }
   },
   {
      "allocations":{
         uuid5(compute1:eth0):{
           "resources":{
               "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
               "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":2000
            },
         },
         uuid5(compute1:eth1):{
           "resources":{
               "NET_BANDWIDTH_EGRESS_KILOBITS_PER_SECOND":1000,
               "NET_BANDWIDTH_INGRESS_KILOBITS_PER_SECOND":1000
            },
         },
         compute_uuid:{
            "resources":{
               "MEMORY_MB":512,
               "DISK_GB":1,
               "VCPU":1
             },
         }
      }
   },
],
"provider_summaries":{
```

(continues on next page)

(continued from previous page)

```
        // unchanged
    }

    "resource_provider-request_group-mappings":[
        {
            "1": [uuid5(compute1:eth0)],
            "2": [uuid5(compute1:eth1)],
            "": [compute_uuid],
        },
        {
            "1": [uuid5(compute1:eth1)],
            "2": [uuid5(compute1:eth0)],
            "": [compute_uuid],
        }
    ]
}
```

This has the advantage that the allocation requests are unchanged and therefore still can be transparently sent back to placement to do the allocation.

This has the disadvantage that one mapping in the `resource_provider-request_group-mappings` connected to one candidate in the allocation_requests list by the list index only.

We decided to go with the primary proposal.

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Upgrade impact

None

Implementation

Assignee(s)

Primary assignee: None

Work Items

- Extend the placement allocation candidate generation algorithm to return the mapping that is internally calculated.
- Extend the API with a new microversion to return the mapping to the API client as well
- Within the same microverison extend the JSON schema for `POST /allocations`, `PUT /allocations/{uuid}`, and `POST /reshaper` to accept (and ignore) the mappings key.

Dependencies

None

Testing

New gabbi tests for the new API microversion and unit test to cover the unhappy path.

Documentation Impact

Placement API ref needs to be updated with the new microversion.

References

History

Table 3: Revisions

Release Name	Description
Stein	Proposed in nova spec repo but was not approved
Train	Re-proposed in the placement repo

In Progress

Support Consumer Types

Include the URL of your story from StoryBoard:

https://storyboard.openstack.org/#!/story/2005473

This spec aims at providing support for services to model `consumer types` in placement. While placement defines a consumer to be an entity consuming resources from a provider it does not provide a way to identify similar "types" of consumers and henceforth allow services to group/query them based on their types. This spec proposes to associate each consumer to a particular type defined by the service owning the consumer.

Problem description

In today's placement world each allocation posted by a service is against a provider for a consumer (ex: for an instance or a migration). However a service may want to distinguish amongst the allocations made against its various types of consumers (ex: nova may want to fetch allocations against instances alone). This is currently not possible in placement and hence the goal is to make placement aware of "types of consumers" for the services.

Use Cases

- Nova using placement as its quota calculation system: Currently this approach uses the nova_api database to calculate the quota on the "number of instances". In order for nova to be able to use placement to count the number of "instance-consumers", there needs to be a way by which we can differentiate "instance-consumers" from "migration-consumers".
- Ironic wanting to differentiate between "standalone-consumer" versus "nova-consumer".

Note that it is not within the scope of placement to model the coordination of the consumer type collisions that may arise between multiple services during their definition. Placement will also not be able to identify or verify correct consumer types (eg, INTANCE versus INSTANCE) from the external service's perspective.

Proposed change

In order to model consumer types in placement, we will add a new `consumer_types` table to the placement database which will have two columns:

1. an `id` which will be of type integer.
2. a `name` which will be of type varchar (maximum of 255 characters) and this will have a unique constraint on it. The pattern restrictions for the name will be similar to placement traits and resource class names, i.e restricted to only `^[A-Z0-9_]+$` with length restrictions being {1, 255}.

A sample look of such a table would be:

id	name
1	UNKNOWN
2	INSTANCE
3	MIGRATION

A new column called `consumer_type_id` would be added to the `consumers` table to map the consumer to its type.

The `POST /allocations` and `PUT /allocations/{consumer_uuid}` REST API's will gain a new (required) key called `consumer_type` which is of type string in their request body's through which the caller can specify what type of consumer it is creating or updating the allocations for. If the specified `consumer_type` key is not present in the `consumer_types` table, a new entry will be created. Also note that once a consumer type is created, it lives on forever. If this becomes a problem in the future for the operators a tool can be provided to clean them up.

In order to maintain parity between the request format of `PUT /allocations/{consumer_uuid}` and response format of `GET /allocations/{consumer_uuid}`, the `consumer_type` key will also be exposed through the response of `GET /allocations/{consumer_uuid}` request.

The external services will be able to leverage this `consumer_type` key through the `GET /usages` REST API which will have a change in the format of its request and response. The request will gain a new optional key called `consumer_type` which will enable users to query usages based on the consumer type. The response will group the resource usages by the specified consumer_type (if consumer_type key is not specified it will return the usages for all the consumer_types) meaning it will gain a new `consumer_type` key. Per consumer type we will also return a `consumer_count` of consumers of that type.

See the *API impact* section for more details on how this would be done.

The above REST API changes and the corresponding changes to the `/reshaper` REST API will be available from a new microversion.

The existing consumers in placement would be mapped to a default consumer type called `UNKNOWN` (which will be the default value while creating the model schema) which means we do not know what type these consumers are and the service to which the consumers belong to needs to update this information if it wants to avail the `consumer_types` feature.

Alternatives

We could create a new REST API to allow users to create consumer types explicitly but it does not make sense to add a new API for a non-user facing feature.

Data model impact

The placement database will get a new `consumer_types` table that will have a default consumer type called `UNKNOWN` and the `consumers` table will get a new `consumer_type_id` column that by default will point to the `UNKNOWN` consumer type. The migration is intended to solely be an alembic migration although a comparision can be done for this versus having a separate online data migration to update null values to "UNKNOWN" to pick the faster one.

API impact

The new `POST /allocations` request will look like this:

```
{
  "30328d13-e299-4a93-a102-61e4ccabe474": {
    "consumer_generation": 1,
    "project_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "user_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "consumer_type": "INSTANCE", # This is new
    "allocations": {
      "e10927c4-8bc9-465d-ac60-d2f79f7e4a00": {
        "resources": {
          "VCPU": 2,
          "MEMORY_MB": 3
        },
        "generation": 4
      }
    }
  },
  "71921e4e-1629-4c5b-bf8d-338d915d2ef3": {
    "consumer_generation": 1,
    "project_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "user_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "consumer_type": "MIGRATION", # This is new
    "allocations": {}
  },
  "48c1d40f-45d8-4947-8d46-52b4e1326df8": {
    "consumer_generation": 1,
    "project_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "user_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "consumer_type": "UNKNOWN", # This is new
    "allocations": {
      "e10927c4-8bc9-465d-ac60-d2f79f7e4a00": {
        "resources": {
          "VCPU": 4,
```

(continues on next page)

(continued from previous page)

```
            "MEMORY_MB": 5
        },
        "generation": 12
      }
    }
  }
}
```

The new `PUT /allocations/{consumer_uuid}` request will look like this:

```
{
  "allocations": {
    "4e061c03-611e-4caa-bf26-999dcff4284e": {
      "resources": {
        "DISK_GB": 20
      }
    },
    "89873422-1373-46e5-b467-f0c5e6acf08f": {
      "resources": {
        "MEMORY_MB": 1024,
        "VCPU": 1
      }
    }
  },
  "consumer_generation": 1,
  "user_id": "66cb2f29-c86d-47c3-8af5-69ae7b778c70",
  "project_id": "42a32c07-3eeb-4401-9373-68a8cdca6784",
  "consumer_type": "INSTANCE" # This is new
}
```

Note that `consumer_type` is a required key for both these requests at this microversion.

The new `GET /usages` response will look like this for a request of type `GET /usages?project_id=<project id>&user_id=<user id>` or `GET /usages?project_id=<project id>` where the consumer_type key is not specified:

```
{
    "usages": {
      "INSTANCE": {
          "consumer_count": 5,
          "DISK_GB": 5,
          "MEMORY_MB": 512,
          "VCPU": 2
      }
      "MIGRATION": {
          "consumer_count": 2,
          "DISK_GB": 5,
          "MEMORY_MB": 512,
          "VCPU": 2
      }
```

(continues on next page)

(continued from previous page)

```
        "UNKNOWN": {
            "consumer_count": 1,
            "DISK_GB": 5,
            "MEMORY_MB": 512,
            "VCPU": 2
        }
    }
}
```

The new `GET /usages` response will look like this for a request of type `GET /usages?project_id=<id>&user_id=<id>&consumer_type="INSTANCE"` or `GET /usages?project_id=<id>&consumer_type="INSTANCE"` where the consumer_type key is specified:

```
{
    "usages": {
      "INSTANCE": {
            "consumer_count": 5,
            "DISK_GB": 5,
            "MEMORY_MB": 512,
            "VCPU": 2
        }
    }
}
```

A special request of the form `GET /usages?project_id=<project id>&consumer_type=all` will be allowed to enabled users to be able to query for the total count of all the consumers. The response for such a request will look like this:

```
{
  "usages": {
      "all": {
            "consumer_count": 3,
            "DISK_GB": 5,
            "MEMORY_MB": 512,
            "VCPU": 2
        }
  }
}
```

Note that `consumer_type` is an optional key for the `GET /usages` request.

The above REST API changes and the corresponding changes to the `/reshaper` REST API will be available from a new microversion.

Security impact

None.

Other end user impact

The external services using this feature like nova should take the responsibility of updating the consumer type of existing consumers from "UNKNOWN" to the actual type through the `PUT /allocations/{consumer_uuid}` REST API.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Upgrade impact

The `placement-manage db sync` command has to be run by the operators in order to upgrade the database schema to accommodate the new changes.

Implementation

Assignee(s)

Primary assignee: <tssurya>

Other contributors: <None>

Work Items

- Add the new `consumer_types` table and create a new `consumer_type_id` column in the `consumers` table with a foreign key constraint to the `id` column of the `consumer_types` table.
- Make the REST API changes in a new microversion for:
  - `POST /allocations`,
  - `PUT /allocations/{consumer_uuid}`,
  - `GET /allocations/{consumer_uuid}`,

- `GET /usages` and
- `/reshaper`

Dependencies

None.

Testing

Unit and functional tests to validate the feature will be added.

Documentation Impact

The placement API reference will be updated to reflect the new changes.

References

History

Table 4: Revisions

Release Name	Description
Train	Introduced

5.1.2 Xena

Implemented

In Progress

Allow provider re-parenting in placement

https://storyboard.openstack.org/#!/story/2008764

This spec proposes to allow re-parenting and un-parenting (or orphaning) RPs via `PUT /resource_providers/{uuid}` API in Placement.

Problem description

Today placement API only allows change the parent of an RP from None to a valid RP UUID. However there are use case when moving an RP between parents make sense.

Use Cases

- An existing PGPU RP needs to be moved under the NUMA RP when NUMA is modeled.
- We have a neutron bug that introduced an unwanted change causing that SRIOV PF RPs was created under the root RP instead of under the neutron agent RP. We can fix the broken logic in neutron but we cannot fix the already wrongly parented RP in the DB via the placement API.

Proposed change

Re-parenting is rejected today and the code has the following comment :

> TODO(jaypipes): For now, "re-parenting" and "un-parenting" are not possible. If the provider already had a parent, we don't allow changing that parent due to various issues, including:
>
> - if the new parent is a descendant of this resource provider, we introduce the possibility of a loop in the graph, which would be very bad
> - potentially orphaning heretofore-descendants
>
> So, for now, let's just prevent re-parenting...

The first reason is moot as the loop check is already needed and implemented for the case when the parent is updated from None to an RP.

The second reason does not make sense to me. By moving an RP under another RP all the descendants should be moved as well. Similarly how the None -> UUID case works today. So I don't see how can we orphan any RP by re-parenting.

I see the following possible cases of move:

- RP moved upwards, downwards, side-wards in the same RP tree
- RP moved to a different tree
- RP moved to top level, becoming a new root RP

From placement perspective every case results in one or more valid RP trees.

Based on the data model if there was allocations against the moved RP those allocations will still refer to the RP after the move. This means that if a consumer has allocation against a single RP tree before the move might have allocation against multiple trees after the RP move. Such consumer is already supported today.

An RP move might invalidate the original intention of the consumer. If the consumer used an allocation candidate query to select and allocate resources then by such query the consumer defined a set of rules (e.g. in_tree, same_subtree) the allocation needs to fulfill. The rules might not be valid after an RP is moved. However placement never promised to keep such invariant as that would require the storage of the rules and correlating allocation candidate queries and allocations. Moreover such issue can already be created with the POST /reshape API as well. Therefore keeping any such invariant is the responsibility

of the client. So I propose to start supporting all form of RP re-parenting in a new placement API microversion.

Alternatives

See the API alternatives below.

Data model impact

None

REST API impact

In a new microversion allow changing the parent_uuid of a resource provider to None or to any valid RP uuid that does not cause a loop in any of the trees via the `PUT /resource_providers/{uuid}` API.

Protecting against unwanted changes

As noted above re-parenting can significantly change the RP model in the Placement database. So such action needs to be done carefully. While the Placement API is already admin only by default, the request is raised on the Xena PTG for extra safety measures against unintentional parent changes. During the spec discussion every the reviewer expressed the view that such safety measure is not really needed. So this spec only propose to use the new microversion and extensive documentation to signal the new behavior.

Still there is the list of alternatives discussed during the review:

- *Do nothing*: While it is considered not safe enough during the PTG, during the spec review we ended up choosing this as the main solution.
- *A new query parameter*: A new query parameter is proposed for the `PUT /resource_providers/{uuid}` API called `allow_reparenting` the default value of the query parameter is `False` and the re-parenting cases defined in this spec is only accepted by Placement if the request contains the new query parameter with the `True`. It is considered hacky to add a query parameter for a PUT request.
- *A new field in the request body*: This new field would have the same meaning as the proposed query parameter but it would be put into the request body. It is considered non-RESTful as such field is not persisted or returned as the result of the PUT request as it does not belong to the representation of the ResourceProvider entity the PUT request updates.
- *A new Header*: Instead of a new query paramtere use a new HTTP header `x-openstack-placement-allow-provider-reparenting:True`. As the name shows this needs a lot more context encoded in it to be specific for the API it modifies while the query parameter already totally API specific.
- *Use a PATCH request for updating the parent*: While this would make the parent change more explicit it would also cause great confusion for the client for multiple reasons:

  1) Other fields of the same resource provider entity can be updated via the PUT request, but not the `parent_uuid` field.

2) Changing the `parent_uuid` field from None to a valid RP uuid is supported by the PUT request but to change it from one RP uuid to another would require a totally different `PATCH` request.

- *Use a sub resource*: Signal the explicit re-parenting either in a form of `PUT /resource-providers/{uuid}/force` or `PUT /resource-providers/{uuid}/parent_uuid/{parent}`. While the second option seems to be acceptable to multiple reviewers, I think it will be confusing similarly to `PATCH`. It would create another way to update a field of an entity while other fields still updated directly on the parent resource.

Security impact

None

Notifications impact

N/A

Other end user impact

None

Performance Impact

The loop detection and the possible update of all the RPs in the changed subtree with a new `root_provider_id` needs extra processing. However the re-parenting operation is considered very infrequent. So the overall Placement performance is not affected.

Other deployer impact

None

Developer impact

None

Upgrade impact

None

Implementation

Assignee(s)

Primary assignee: balazs-gibizer

Feature Liaison

Feature liaison: None

Work Items

- Add a new microversion to the Placement API. Implement an extended loop detection and update `root_provider_id` of the subtree if needed.
- Mark the new microversion osc-placement as supported.

Dependencies

None

Testing

- Unit testing
- Gabbit API testing

Documentation Impact

- API doc needs to be updated. Warn the user that this is a potentially dangerous operation.

References

None

History

Table 5: Revisions

Release Name	Description
Xena	Introduced

Support Consumer Types

https://storyboard.openstack.org/#!/story/2005473

This spec aims at providing support for services to model `consumer types` in placement. While placement defines a consumer to be an entity consuming resources from a provider it does not provide a way to identify similar "types" of consumers and henceforth allow services to group/query them based on their types. This spec proposes to associate each consumer to a particular type defined by the service owning the consumer.

Problem description

In today's placement world each allocation posted by a service is against a provider for a consumer (ex: for an instance or a migration). However a service may want to distinguish amongst the allocations made against its various types of consumers (ex: nova may want to fetch allocations against instances alone). This is currently not possible in placement and hence the goal is to make placement aware of "types of consumers" for the services.

Use Cases

- Nova using placement as its quota calculation system: Currently this approach uses the nova_api database to calculate the quota on the "number of instances". In order for nova to be able to use placement to count the number of "instance-consumers", there needs to be a way by which we can differentiate "instance-consumers" from "migration-consumers".
- Ironic wanting to differentiate between "standalone-consumer" versus "nova-consumer".

Note that it is not within the scope of placement to model the coordination of the consumer type collisions that may arise between multiple services during their definition. Placement will also not be able to identify or verify correct consumer types (eg, INTANCE versus INSTANCE) from the external service's perspective.

Proposed change

In order to model consumer types in placement, we will add a new `consumer_types` table to the placement database which will have two columns:

1. an `id` which will be of type integer.
2. a `name` which will be of type varchar (maximum of 255 characters) and this will have a unique constraint on it. The pattern restrictions for the name will be similar to placement traits and resource class names, i.e restricted to only `^[A-Z0-9_]+$` with length restrictions being {1, 255}.

A sample look of such a table would be:

id	name
1	INSTANCE
2	MIGRATION

A new column called `consumer_type_id` would be added to the `consumers` table to map the consumer to its type.

The `POST /allocations` and `PUT /allocations/{consumer_uuid}` REST API's will gain a new (required) key called `consumer_type` which is of type string in their request body's through which the caller can specify what type of consumer it is creating or updating the allocations for. If the specified `consumer_type` key is not present in the `consumer_types` table, a new entry will be created. Also note that once a consumer type is created, it lives on forever. If this becomes a problem in the future for the operators a tool can be provided to clean them up.

In order to maintain parity between the request format of `PUT /allocations/{consumer_uuid}` and response format of `GET /allocations/{consumer_uuid}`, the `consumer_type` key will also be exposed through the response of `GET /allocations/{consumer_uuid}` request.

The external services will be able to leverage this `consumer_type` key through the `GET /usages` REST API which will have a change in the format of its request and response. The request will gain a new optional key called `consumer_type` which will enable users to query usages based on the consumer type. The response will group the resource usages by the specified consumer_type (if consumer_type key is not specified it will return the usages for all the consumer_types) meaning it will gain a new `consumer_type` key. Per consumer type we will also return a `consumer_count` of consumers of that type.

See the *REST API impact* section for more details on how this would be done.

The above REST API changes and the corresponding changes to the `/reshaper` REST API will be available from a new microversion.

The existing consumers in placement will have a `NULL` value in their consumer_type_id field, which means we do not know what type these consumers are and the service to which the consumers belong to needs to update this information if it wants to avail the `consumer_types` feature.

Alternatives

We could create a new REST API to allow users to create consumer types explicitly but it does not make sense to add a new API for a non-user facing feature.

Data model impact

The placement database will get a new `consumer_types` table and the `consumers` table will get a new `consumer_type_id` column that by default will be `NULL`.

REST API impact

The new `POST /allocations` request will look like this:

```
{
  "30328d13-e299-4a93-a102-61e4ccabe474": {
    "consumer_generation": 1,
    "project_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "user_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "consumer_type": "INSTANCE", # This is new
    "allocations": {
      "e10927c4-8bc9-465d-ac60-d2f79f7e4a00": {
```

(continues on next page)

(continued from previous page)

```
        "resources": {
          "VCPU": 2,
          "MEMORY_MB": 3
        },
        "generation": 4
      }
    }
  },
  "71921e4e-1629-4c5b-bf8d-338d915d2ef3": {
    "consumer_generation": 1,
    "project_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "user_id": "131d4efb-abc0-4872-9b92-8c8b9dc4320f",
    "consumer_type": "MIGRATION", # This is new
    "allocations": {}
  }
}
```

The new `PUT /allocations/{consumer_uuid}` request will look like this:

```
{
  "allocations": {
    "4e061c03-611e-4caa-bf26-999dcff4284e": {
      "resources": {
        "DISK_GB": 20
      }
    },
    "89873422-1373-46e5-b467-f0c5e6acf08f": {
      "resources": {
        "MEMORY_MB": 1024,
        "VCPU": 1
      }
    }
  },
  "consumer_generation": 1,
  "user_id": "66cb2f29-c86d-47c3-8af5-69ae7b778c70",
  "project_id": "42a32c07-3eeb-4401-9373-68a8cdca6784",
  "consumer_type": "INSTANCE" # This is new
}
```

Note that `consumer_type` is a required key for both these requests at this microversion.

The new `GET /usages` response will look like this for a request of type `GET /usages?project_id=<project id>&user_id=<user id>` or `GET /usages?project_id=<project id>` where the consumer_type key is not specified:

```
{
    "usages": {
      "INSTANCE": {
          "consumer_count": 5,
          "DISK_GB": 5,
```

(continues on next page)

(continued from previous page)

```
            "MEMORY_MB": 512,
            "VCPU": 2
        }
        "MIGRATION": {
            "consumer_count": 2,
            "DISK_GB": 5,
            "MEMORY_MB": 512,
            "VCPU": 2
        }
        "unknown": {
            "consumer_count": 1,
            "DISK_GB": 5,
            "MEMORY_MB": 512,
            "VCPU": 2
        }
    }
}
```

The new `GET /usages` response will look like this for a request of type `GET /usages?project_id=<id>&user_id=<id>&consumer_type="INSTANCE"` or `GET /usages?project_id=<id>&consumer_type="INSTANCE"` where the consumer_type key is specified:

```
{
    "usages": {
      "INSTANCE": {
            "consumer_count": 5,
            "DISK_GB": 5,
            "MEMORY_MB": 512,
            "VCPU": 2
        }
    }
}
```

A special request of the form `GET /usages?project_id=<project id>&consumer_type=all` will be allowed to enable users to be able to query for the total count of all the consumers. The response for such a request will look like this:

```
{
  "usages": {
      "all": {
          "consumer_count": 3,
          "DISK_GB": 5,
          "MEMORY_MB": 512,
          "VCPU": 2
      }
  }
}
```

A special request of the form `GET /usages?project_id=<project id>&consumer_type=unknown` will be allowed to enable users to be able to query for the total count of the consumers that have no consumer type assigned. The response for such a request will look like this:

```
{
  "usages": {
      "unknown": {
          "consumer_count": 3,
          "DISK_GB": 5,
          "MEMORY_MB": 512,
          "VCPU": 2
      }
  }
}
```

Note that `consumer_type` is an optional key for the `GET /usages` request.

The above REST API changes and the corresponding changes to the `/reshaper` REST API will be available from a new microversion.

Security impact

None.

Notifications impact

N/A

Other end user impact

The external services using this feature like nova should take the responsibility of updating the consumer type of existing consumers from `NULL` to the actual type through the `PUT /allocations/{consumer_uuid}` REST API.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

Upgrade impact

The `placement-manage db sync` command has to be run by the operators in order to upgrade the database schema to accommodate the new changes.

Implementation

Assignee(s)

Primary assignee: <melwitt>

Other contributors: <tssurya> <cdent>

Work Items

- Add the new `consumer_types` table and create a new `consumer_type_id` column in the `consumers` table with a foreign key constraint to the `id` column of the `consumer_types` table.
- Make the REST API changes in a new microversion for:
  - `POST /allocations`,
  - `PUT /allocations/{consumer_uuid}`,
  - `GET /allocations/{consumer_uuid}`,
  - `GET /usages` and
  - `/reshaper`

Dependencies

None.

Testing

Unit and functional tests to validate the feature will be added.

Documentation Impact

The placement API reference will be updated to reflect the new changes.

References

History

Table 6: Revisions

Release Name	Description
Train	Introduced
Xena	Reproposed

Example Spec - The title

Include the URL of your story from StoryBoard:

https://storyboard.openstack.org/#!/story/XXXXXXX

Introduction paragraph -- why are we doing anything? A single paragraph of prose that operators can understand. The title and this first paragraph should be used as the subject line and body of the commit message respectively.

Some notes about the spec process:

- Not all blueprints need a spec, start with a story.
- The aim of this document is first to define the problem we need to solve, and second agree the overall approach to solve that problem.
- This is not intended to be extensive documentation for a new feature. For example, there is no need to specify the exact configuration changes, nor the exact details of any DB model changes. But you should still define that such changes are required, and be clear on how that will affect upgrades.
- You should aim to get your spec approved before writing your code. While you are free to write prototypes and code before getting your spec approved, its possible that the outcome of the spec review process leads you towards a fundamentally different solution than you first envisaged.
- But API changes are held to a much higher level of scrutiny. As soon as an API change merges, we must assume it could be in production somewhere, and as such, we then need to support that API change forever. To avoid getting that wrong, we do want lots of details about API changes up front.

Some notes about using this template:

- Your spec should be in ReSTructured text, like this template.
- Please wrap text at 79 columns.
- The filename in the git repository should start with the StoryBoard story number. For example: `2005171-allocation-partitioning.rst`.
- Please do not delete any of the sections in this template. If you have nothing to say for a whole section, just write: None

- For help with syntax, see http://sphinx-doc.org/rest.html
- To test out your formatting, build the docs using `tox -e docs` and see the generated HTML file in doc/build/html/specs/<path_of_your_file>. The generated file will have an `.html` extension where the original has `.rst`.
- If you would like to provide a diagram with your spec, ascii diagrams are often the best choice. http://asciiflow.com/ is a useful tool. If ascii is insufficient, you have the option to use seqdiag or actdiag.

Problem description

A detailed description of the problem. What problem is this feature addressing?

Use Cases

What use cases does this address? What impact on actors does this change have? Ensure you are clear about the actors in each use case: Developer, End User, Deployer etc.

Proposed change

Here is where you cover the change you propose to make in detail. How do you propose to solve this problem?

If this is one part of a larger effort make it clear where this piece ends. In other words, what's the scope of this effort?

At this point, if you would like to get feedback on if the problem and proposed change fit in placement, you can stop here and post this for review saying: Posting to get preliminary feedback on the scope of this spec.

Alternatives

What other ways could we do this thing? Why aren't we using those? This doesn't have to be a full literature review, but it should demonstrate that thought has been put into why the proposed solution is an appropriate one.

Data model impact

Changes which require modifications to the data model often have a wider impact on the system. The community often has strong opinions on how the data model should be evolved, from both a functional and performance perspective. It is therefore important to capture and gain agreement as early as possible on any proposed changes to the data model.

Questions which need to be addressed by this section include:

- What new data objects and/or database schema changes is this going to require?
- What database migrations will accompany this change?

- How will the initial set of new data objects be generated? For example if you need to take into account existing instances, or modify other existing data, describe how that will work.

API impact

Each API method which is either added or changed should have the following

- Specification for the method
  - A description of what the method does suitable for use in user documentation
  - Method type (POST/PUT/GET/DELETE)
  - Normal http response code(s)
  - Expected error http response code(s)
    - A description for each possible error code should be included describing semantic errors which can cause it such as inconsistent parameters supplied to the method, or when a resource is not in an appropriate state for the request to succeed. Errors caused by syntactic problems covered by the JSON schema definition do not need to be included.
  - URL for the resource
    - URL should not include underscores; use hyphens instead.
  - Parameters which can be passed via the url
  - JSON schema definition for the request body data if allowed
    - Field names should use snake_case style, not camelCase or MixedCase style.
  - JSON schema definition for the response body data if any
    - Field names should use snake_case style, not camelCase or MixedCase style.
- Example use case including typical API samples for both data supplied by the caller and the response
- Discuss any policy changes, and discuss what things a deployer needs to think about when defining their policy.

Note that the schema should be defined as restrictively as possible. Parameters which are required should be marked as such and only under exceptional circumstances should additional parameters which are not defined in the schema be permitted (eg additionaProperties should be False).

Reuse of existing predefined parameter types such as regexps for passwords and user defined names is highly encouraged.

Security impact

Describe any potential security impact on the system. Some of the items to consider include:

- Does this change touch sensitive data such as tokens, keys, or user data?
- Does this change alter the API in a way that may impact security, such as a new way to access sensitive information or a new way to log in?
- Does this change involve cryptography or hashing?
- Does this change require the use of sudo or any elevated privileges?
- Does this change involve using or parsing user-provided data? This could be directly at the API level or indirectly such as changes to a cache layer.
- Can this change enable a resource exhaustion attack, such as allowing a single API interaction to consume significant server resources? Some examples of this include launching subprocesses for each connection, or entity expansion attacks in XML.

For more detailed guidance, please see the OpenStack Security Guidelines as a reference (https://wiki.openstack.org/wiki/Security/Guidelines). These guidelines are a work in progress and are designed to help you identify security best practices. For further information, feel free to reach out to the OpenStack Security Group at openstack-security@lists.openstack.org.

Other end user impact

Aside from the API, are there other ways a user will interact with this feature?

- Does this change have an impact on osc-placement? What does the user interface there look like?

Performance Impact

Describe any potential performance impact on the system, for example how often will new code be called, and is there a major change to the calling pattern of existing code.

Examples of things to consider here include:

- A small change in a utility function or a commonly used decorator can have a large impacts on performance.
- Calls which result in a database queries can have a profound impact on performance when called in critical sections of the code.
- Will the change include any locking, and if so what considerations are there on holding the lock?

Other deployer impact

Discuss things that will affect how you deploy and configure OpenStack that have not already been mentioned, such as:

- What config options are being added? Should they be more generic than proposed? Are the default values ones which will work well in real deployments?
- Is this a change that takes immediate effect after its merged, or is it something that has to be explicitly enabled?
- If this change is a new binary, how would it be deployed?
- Please state anything that those doing continuous deployment, or those upgrading from the previous release, need to be aware of. Also describe any plans to deprecate configuration values or features.

Developer impact

Discuss things that will affect other developers working on OpenStack.

Upgrade impact

Describe any potential upgrade impact on the system.

Implementation

Assignee(s)

Who is leading the writing of the code? Or is this a blueprint where you're throwing it out there to see who picks it up?

If more than one person is working on the implementation, please designate the primary author and contact.

Primary assignee: <IRC nick or None>

Other contributors: <IRC nick or None>

Work Items

Work items or tasks -- break the feature up into the things that need to be done to implement it. Those parts might end up being done by different people, but we're mostly trying to understand the timeline for implementation.

Dependencies

- Include specific references to other specs or stories that this one either depends on or is related to.
- If this requires new functionality in another project that is not yet used document that fact.
- Does this feature require any new library dependencies or code otherwise not included in OpenStack? Or does it depend on a specific version of a library?

Testing

Please discuss the important scenarios that need to be tested, as well as specific edge cases we should be ensuring work correctly.

Documentation Impact

Which audiences are affected most by this change, and which documentation titles on docs.openstack.org should be updated because of this change? Don't repeat details discussed above, but reference them here in the context of documentation for multiple audiences.

References

Please add any useful references here. You are not required to have any references. Moreover, this specification should still make sense when your references are unavailable. Examples of what you could include are:

- Links to mailing list or IRC discussions
- Links to notes from a summit session
- Links to relevant research, if appropriate
- Anything else you feel it is worthwhile to refer to

History

Optional section intended to be used each time the spec is updated to describe new design, API or any database schema updated. Useful to let the reader understand how the spec has changed over time.

Table 7: Revisions

Release Name	Description
<Replace With Current Release>	Introduced

CHAPTER

SIX

DEPLOYMENT

6.1 Installation

Note: Before the Stein release the placement code was in Nova alongside the compute REST API code (nova-api). Make sure that the release version of this document matches the release version you want to deploy.

6.1.1 Steps Overview

This subsection gives an overview of the process without going into detail on the methods used.

1. Deploy the API service

Placement provides a `placement-api` WSGI script for running the service with Apache, nginx or other WSGI-capable web servers. Depending on what packaging solution is used to deploy OpenStack, the WSGI script may be in `/usr/bin` or `/usr/local/bin`.

`placement-api`, as a standard WSGI script, provides a module level `application` attribute that most WSGI servers expect to find. This means it is possible to run it with lots of different servers, providing flexibility in the face of different deployment scenarios. Common scenarios include:

- apache2 with mod_wsgi
- apache2 with mod_proxy_uwsgi
- nginx with uwsgi
- nginx with gunicorn

In all of these scenarios the host, port and mounting path (or prefix) of the application is controlled in the web server's configuration, not in the configuration (`placement.conf`) of the placement application.

When placement was first added to DevStack it used the `mod_wsgi` style. Later it was updated to use mod_proxy_uwsgi. Looking at those changes can be useful for understanding the relevant options.

DevStack is configured to host placement at `/placement` on either the default port for http or for https (`80` or `443`) depending on whether TLS is being used. Using a default port is desirable.

By default, the placement application will get its configuration for settings such as the database connection URL from `/etc/placement/placement.conf`. The directory the configuration file will be found in can be changed by setting `OS_PLACEMENT_CONFIG_DIR` in the environment of the process that starts the application. With recent releases of `oslo.config`, configuration options may also be set in the environment.

Note: When using uwsgi with a front end (e.g., apache2 or nginx) something needs to ensure that the uwsgi process is running. In DevStack this is done with systemd. This is one of many different ways to manage uwsgi.

This document refrains from declaring a set of installation instructions for the placement service. This is because a major point of having a WSGI application is to make the deployment as flexible as possible. Because the placement API service is itself stateless (all state is in the database), it is possible to deploy as many servers as desired behind a load balancing solution for robust and simple scaling. If you familiarize yourself with installing generic WSGI applications (using the links in the common scenarios list, above), those techniques will be applicable here.

2. Synchronize the database

The placement service uses its own database, defined in the `placement_database` section of configuration. The `placement_database.connection` option **must** be set or the service will not start. The command line tool *placement-manage* can be used to migrate the database tables to their correct form, including creating them. The database described by the `connection` option must already exist and have appropriate access controls defined.

Another option for synchronization is to set `placement_database.sync_on_startup` to `True` in configuration. This will perform any missing database migrations as the placement web service starts. Whether you choose to sync automaticaly or use the command line tool depends on the constraints of your environment and deployment tooling.

Warning: In the Stein release, the placement code was extracted from nova. If you are upgrading to use the extracted placement you will need to migrate your placement data from the `nova_api` database to the `placement` database. You can find sample scripts that may help with this in the placement repository: mysql-migrate-db.sh and postgresql-migrate-db.sh. See also *Upgrade Notes*.

Note: Upgrading to the extracted placement at the same time as the other OpenStack services when upgrading to Stein is an option but *is not required*. The nova code will continue to have a copy of the placement service in its Stein release. However this copy **will** be deleted in Train and switching to the extracted version before upgrading to Train (potentially with the help of the scripts above) will be required.

3. Create accounts and update the service catalog

Create a **placement** service user with an **admin** role in Keystone.

The placement API is a separate service and thus should be registered under a **placement** service type in the service catalog. Clients of placement, such as the resource tracker in the nova-compute node, will use the service catalog to find the placement endpoint.

See *Configure User and Endpoints* for examples of creating the service user and catalog entries.

Devstack sets up the placement service on the default HTTP port (80) with a `/placement` prefix instead of using an independent port.

6.1.2 Installation Packages

This section provides instructions on installing placement from Linux distribution packages.

> **Warning:** These installation documents are a work in progress. Some of the distribution packages mentioned are not yet available so the instructions **will not work**.

The placement service provides an HTTP API used to track resource provider inventories and usages. More detail can be found at the *placement overview*.

Placement operates as a web service over a data model. Installation involves creating the necessary database and installing and configuring the web service. This is a straightforward process, but there are quite a few steps to integrate placement with the rest of an OpenStack cloud.

Note: Placement is required by some of the other OpenStack services, notably nova, therefore it should be installed before those other services but after Identity (keystone).

Install and configure Placement from PyPI

The section describes how to install and configure the placement service using packages from PyPI. Placement works with Python version 2.7, but version 3.6 or higher is recommended.

This document assumes you have a working MySQL server and a working Python environment, including the *pip* package installer. Depending on your environment, you may wish to install placement in a virtualenv.

This document describes how to run placement with uwsgi as its web server. This is but one of many different ways to host the service. Placement is a well-behaved WSGI application so should be straightforward to host with any WSGI server.

If using placement in an OpenStack environment, you will need to ensure it is up and running before starting services that use it but after services it uses. That means after Keystone, but before anything else.

Prerequisites

Before installing the service, you will need to create the database, service credentials, and API endpoints, as described in the following sections.

pip

Install pip from PyPI.

Note: Examples throughout this reference material use the `pip` command. This may need to be pathed or spelled differently (e.g. `pip3`) depending on your installation and Python version.

---

python-openstackclient

If not already installed, install the `openstack` command line tool:

```
# pip install python-openstackclient
```

Create Database

Placement is primarily tested with MySQL/MariaDB so that is what is described here. It also works well with PostgreSQL and likely with many other databases supported by sqlalchemy.

To create the database, complete these steps:

1. Use the database access client to connect to the database server as the `root` user or by using `sudo` as appropriate:

   ```
   # mysql
   ```

2. Create the `placement` database:

   ```
   MariaDB [(none)]> CREATE DATABASE placement;
   ```

3. Grant proper access to the database:

   ```
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
   ↪'localhost' \
     IDENTIFIED BY 'PLACEMENT_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@'%' \
     IDENTIFIED BY 'PLACEMENT_DBPASS';
   ```

   Replace `PLACEMENT_DBPASS` with a suitable password.

4. Exit the database access client.

Configure User and Endpoints

Note: If you are not using Keystone, you can skip the steps below but will need to configure the *api.auth_strategy* setting with a value of `noauth2`. See also *Quick Placement Development*.

Note: You will need to authenticate to Keystone as an `admin` before making these calls. There are many different ways to do this, depending on how your system was set up. If you do not have an `admin-openrc` file, you will have something similar.

Important: These documents use an endpoint URL of `http://controller:8778/` as an example only. You should configure placement to use whatever hostname and port works best for your environment. Using SSL on the default port, with either a domain or path specific to placement, is recommended. For example: `https://mygreatcloud.com/placement` or `https://placement.mygreatcloud.com/`.

1. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

2. Create a Placement service user using your chosen `PLACEMENT_PASS`:

```
$ openstack user create --domain default --password-prompt placement

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | fa742015a6494a949f67629884fc7ec8 |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

3. Add the Placement user to the service project with the admin role:

```
$ openstack role add --project service --user placement admin
```

Note: This command provides no output.

4. Create the Placement API entry in the service catalog:

```
$ openstack service create --name placement \
  --description "Placement API" placement

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Placement API                    |
| enabled     | True                             |
| id          | 2d1a27022e6e4185b86adac4444c495f |
| name        | placement                        |
| type        | placement                        |
+-------------+----------------------------------+
```

5. Create the Placement API service endpoints:

Note: Depending on your environment, the URL for the endpoint will vary by port (possibly 8780 instead of 8778, or no port at all) and hostname. You are responsible for determining the correct URL.

```
$ openstack endpoint create --region RegionOne \
  placement public http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 2b1b2637908b4137a9c2e0470487cbc0 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement internal http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 02bcda9a150a4bd7993ff4879df971ab |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
```

(continues on next page)

(continued from previous page)

```
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement admin http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d71177b9e0f406f98cbff198d74b182 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
```

Install and configure components

The default location of the placement configuration file is `/etc/placement/placement.conf`. A different directory may be chosen by setting `OS_PLACEMENT_CONFIG_DIR` in the environment. It is also possible to run the service with a partial or no configuration file and set some options in the environment. See *Configuration Guide* for additional configuration settings not mentioned here.

Note: In the steps below, `controller` is used as a stand in for the hostname of the hosts where keystone, mysql, and placement are running. These may be distinct. The keystone host (used for `auth_url` and `www_authenticate_uri`) should be the unversioned public endpoint for the Identity service.

---

1. Install placement and required database libraries:

```
# pip install openstack-placement pymysql
```

2. Create the `/etc/placement/placement.conf` file and complete the following actions:

   - Create a `[placement_database]` section and configure database access:

```
[placement_database]
connection = mysql+pymysql://placement:PLACEMENT_DBPASS@controller/
↪placement
```

     Replace `PLACEMENT_DBPASS` with the password you chose for the placement database.

   - Create `[api]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[api]
auth_strategy = keystone  # use noauth2 if not using keystone

[keystone_authtoken]
www_authenticate_uri = http://controller:5000/
auth_url = http://controller:5000/
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = PLACEMENT_PASS
```

Replace `PLACEMENT_PASS` with the password you chose for the `placement` user in the Identity service.

Note: The value of `user_name`, `password`, `project_domain_name` and `user_domain_name` need to be in sync with your keystone config.

- You may wish to set the *debug* option to `True` to produce more verbose log output.

3. Populate the `placement` database:

```
$ placement-manage db sync
```

Note: An alternative is to use the *placement_database.sync_on_startup* option.

Finalize installation

Now that placement itself has been installed we need to launch the service in a web server. What follows provides a very basic web server that, while relatively performant, is not set up to be easy to manage. Since there are many web servers and many ways to manage them, such things are outside the scope of this document.

Install and run the web server:

1. Install the `uwsgi` package (these instructions are against version 2.0.18):

```
# pip install uwsgi
```

2. Run the server with the placement WSGI application in a terminal window:

Warning: Make sure you are using the correct `uwsgi` binary. It may be in multiple places in your path. The wrong version will fail and complain about bad arguments.

```
# uwsgi -M --http :8778 --wsgi-file /usr/local/bin/placement-api \
        --processes 2 --threads 10
```

3. In another terminal confirm the server is running using `curl`. The URL should match the public endpoint set in *Configure User and Endpoints*.

```
$ curl http://controller:8778/
```

The output will look something like this:


```
{
   "versions" : [
      {
         "id" : "v1.0",
         "max_version" : "1.31",
         "links" : [
            {
               "href" : "",
               "rel" : "self"
            }
         ],
         "min_version" : "1.0",
         "status" : "CURRENT"
      }
   ]
}
```


Further interactions with the system can be made with osc-placement.

Install and configure Placement for openSUSE and SUSE Linux Enterprise

This section describes how to install and configure the placement service when using openSUSE or SUSE Linux Enterprise packages.

Prerequisites

Before you install and configure the placement service, you must create a database, service credentials, and API endpoints.

Create Database

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `placement` database:

```
MariaDB [(none)]> CREATE DATABASE placement;
```

- Grant proper access to the database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
↪'localhost' \
  IDENTIFIED BY 'PLACEMENT_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
↪'%' \
  IDENTIFIED BY 'PLACEMENT_DBPASS';
```

Replace `PLACEMENT_DBPASS` with a suitable password.

- Exit the database access client.

Configure User and Endpoints

1. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

2. Create a Placement service user using your chosen `PLACEMENT_PASS`:

```
$ openstack user create --domain default --password-prompt placement

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | fa742015a6494a949f67629884fc7ec8 |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

3. Add the Placement user to the service project with the admin role:

```
$ openstack role add --project service --user placement admin
```

Note: This command provides no output.

4. Create the Placement API entry in the service catalog:

```
$ openstack service create --name placement \
  --description "Placement API" placement
```

(continues on next page)

(continued from previous page)

```
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Placement API                    |
| enabled     | True                             |
| id          | 2d1a27022e6e4185b86adac4444c495f |
| name        | placement                        |
| type        | placement                        |
+-------------+----------------------------------+
```

5. Create the Placement API service endpoints:

Note: Depending on your environment, the URL for the endpoint will vary by port (possibly 8780 instead of 8778, or no port at all) and hostname. You are responsible for determining the correct URL.

```
$ openstack endpoint create --region RegionOne \
  placement public http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 2b1b2637908b4137a9c2e0470487cbc0 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement internal http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 02bcda9a150a4bd7993ff4879df971ab |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
```

(continues on next page)

(continued from previous page)

```
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement admin http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d71177b9e0f406f98cbff198d74b182 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
```

Install and configure components

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

Note: As of the Newton release, SUSE OpenStack packages are shipped with the upstream default configuration files. For example, `/etc/placement/placement.conf` has customizations in `/etc/placement/placement.conf.d/010-placement.conf`. While the following instructions modify the default configuration file, adding a new file in `/etc/placement/placement.conf.d` achieves the same result.

1. Install the packages:

```
# zypper install openstack-placement
```

2. Edit the `/etc/placement/placement.conf` file and complete the following actions:

   - In the `[placement_database]` section, configure database access:

```
[placement_database]
# ...
connection = mysql+pymysql://placement:PLACEMENT_DBPASS@controller/
↪placement
```

     Replace `PLACEMENT_DBPASS` with the password you chose for the placement database.

   - In the `[api]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[api]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
auth_url = http://controller:5000/v3
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = PLACEMENT_PASS
```

Replace `PLACEMENT_PASS` with the password you chose for the `placement` user in the Identity service.

Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

Note: The value of `user_name`, `password`, `project_domain_name` and `user_domain_name` need to be in sync with your keystone config.

3. Populate the `placement` database:

```
# su -s /bin/sh -c "placement-manage db sync" placement
```

Note: Ignore any deprecation messages in this output.

Finalize installation

- Enable the placement API Apache vhost:

```
# mv /etc/apache2/vhosts.d/openstack-placement-api.conf.sample \
  /etc/apache2/vhosts.d/openstack-placement-api.conf
# systemctl reload apache2.service
```

Install and configure Placement for Red Hat Enterprise Linux and CentOS

This section describes how to install and configure the placement service when using Red Hat Enterprise Linux or CentOS packages.

Prerequisites

Before you install and configure the placement service, you must create a database, service credentials, and API endpoints.

Create Database

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `placement` database:

     ```
     MariaDB [(none)]> CREATE DATABASE placement;
     ```

   - Grant proper access to the database:

     ```
     MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
     ↪'localhost' \
       IDENTIFIED BY 'PLACEMENT_DBPASS';
     MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
     ↪'%' \
       IDENTIFIED BY 'PLACEMENT_DBPASS';
     ```

     Replace `PLACEMENT_DBPASS` with a suitable password.

   - Exit the database access client.

Configure User and Endpoints

1. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

2. Create a Placement service user using your chosen `PLACEMENT_PASS`:

   ```
   $ openstack user create --domain default --password-prompt placement

   User Password:
   Repeat User Password:
   +---------------------+----------------------------------+
   | Field               | Value                            |
   +---------------------+----------------------------------+
   ```

(continues on next page)

(continued from previous page)

```
| domain_id           | default                          |
| enabled             | True                             |
| id                  | fa742015a6494a949f67629884fc7ec8 |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

3. Add the Placement user to the service project with the admin role:

```
$ openstack role add --project service --user placement admin
```

Note: This command provides no output.

4. Create the Placement API entry in the service catalog:

```
$ openstack service create --name placement \
  --description "Placement API" placement

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Placement API                    |
| enabled     | True                             |
| id          | 2d1a27022e6e4185b86adac4444c495f |
| name        | placement                        |
| type        | placement                        |
+-------------+----------------------------------+
```

5. Create the Placement API service endpoints:

Note: Depending on your environment, the URL for the endpoint will vary by port (possibly 8780 instead of 8778, or no port at all) and hostname. You are responsible for determining the correct URL.

```
$ openstack endpoint create --region RegionOne \
  placement public http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 2b1b2637908b4137a9c2e0470487cbc0 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
```

(continues on next page)

(continued from previous page)

```
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement internal http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 02bcda9a150a4bd7993ff4879df971ab |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement admin http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d71177b9e0f406f98cbff198d74b182 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
```

Install and configure components

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# yum install openstack-placement-api
```

2. Edit the `/etc/placement/placement.conf` file and complete the following actions:

   - In the `[placement_database]` section, configure database access:

     ```
     [placement_database]
     # ...
     connection = mysql+pymysql://placement:PLACEMENT_DBPASS@controller/
     ↪placement
     ```

     Replace `PLACEMENT_DBPASS` with the password you chose for the placement database.

   - In the `[api]` and `[keystone_authtoken]` sections, configure Identity service access:

     ```
     [api]
     # ...
     auth_strategy = keystone

     [keystone_authtoken]
     # ...
     auth_url = http://controller:5000/v3
     memcached_servers = controller:11211
     auth_type = password
     project_domain_name = Default
     user_domain_name = Default
     project_name = service
     username = placement
     password = PLACEMENT_PASS
     ```

     Replace `PLACEMENT_PASS` with the password you chose for the `placement` user in the Identity service.

     Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

     Note: The value of `user_name`, `password`, `project_domain_name` and `user_domain_name` need to be in sync with your keystone config.

3. Populate the `placement` database:

   ```
   # su -s /bin/sh -c "placement-manage db sync" placement
   ```

   Note: Ignore any deprecation messages in this output.

Finalize installation

- Restart the httpd service:

```
# systemctl restart httpd
```

Install and configure Placement for Ubuntu

This section describes how to install and configure the placement service when using Ubuntu packages.

Prerequisites

Before you install and configure the placement service, you must create a database, service credentials, and API endpoints.

Create Database

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

```
# mysql
```

   - Create the `placement` database:

```
MariaDB [(none)]> CREATE DATABASE placement;
```

   - Grant proper access to the database:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
↪'localhost' \
  IDENTIFIED BY 'PLACEMENT_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@
↪'%' \
  IDENTIFIED BY 'PLACEMENT_DBPASS';
```

     Replace `PLACEMENT_DBPASS` with a suitable password.

   - Exit the database access client.

Configure User and Endpoints

1. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

2. Create a Placement service user using your chosen `PLACEMENT_PASS`:

```
$ openstack user create --domain default --password-prompt placement

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | fa742015a6494a949f67629884fc7ec8 |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

3. Add the Placement user to the service project with the admin role:

```
$ openstack role add --project service --user placement admin
```

Note: This command provides no output.

4. Create the Placement API entry in the service catalog:

```
$ openstack service create --name placement \
  --description "Placement API" placement

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Placement API                    |
| enabled     | True                             |
| id          | 2d1a27022e6e4185b86adac4444c495f |
| name        | placement                        |
| type        | placement                        |
+-------------+----------------------------------+
```

5. Create the Placement API service endpoints:

Note: Depending on your environment, the URL for the endpoint will vary by port (possibly 8780 instead of 8778, or no port at all) and hostname. You are responsible for determining the correct URL.

```
$ openstack endpoint create --region RegionOne \
  placement public http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
```

(continues on next page)

(continued from previous page)

```
| enabled      | True                             |
| id           | 2b1b2637908b4137a9c2e0470487cbc0 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement internal http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 02bcda9a150a4bd7993ff4879df971ab |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  placement admin http://controller:8778

+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d71177b9e0f406f98cbff198d74b182 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 2d1a27022e6e4185b86adac4444c495f |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
```

Install and configure components

Note: Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# apt install placement-api
```

2. Edit the `/etc/placement/placement.conf` file and complete the following actions:

   - In the `[placement_database]` section, configure database access:

```
[placement_database]
# ...
connection = mysql+pymysql://placement:PLACEMENT_DBPASS@controller/
↪placement
```

     Replace `PLACEMENT_DBPASS` with the password you chose for the placement database.

   - In the `[api]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[api]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
auth_url = http://controller:5000/v3
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = PLACEMENT_PASS
```

     Replace `PLACEMENT_PASS` with the password you chose for the `placement` user in the Identity service.

     Note: Comment out or remove any other options in the `[keystone_authtoken]` section.

     Note: The value of `user_name`, `password`, `project_domain_name` and `user_domain_name` need to be in sync with your keystone config.

3. Populate the `placement` database:

```
# su -s /bin/sh -c "placement-manage db sync" placement
```

Note: Ignore any deprecation messages in this output.

Finalize installation

- Reload the web server to adjust to get new configuration settings for placement.

```
# service apache2 restart
```

Verify Installation

Verify operation of the placement service.

Note: You will need to authenticate to the identity service as an `admin` before making these calls. There are many different ways to do this, depending on how your system was set up. If you do not have an `admin-openrc` file, you will have something similar.

1. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

2. Perform status checks to make sure everything is in order:

```
$ placement-status upgrade check
+----------------------------------+
| Upgrade Check Results            |
+----------------------------------+
| Check: Missing Root Provider IDs |
| Result: Success                  |
| Details: None                    |
+----------------------------------+
| Check: Incomplete Consumers      |
| Result: Success                  |
| Details: None                    |
+----------------------------------+
```

   The output of that command will vary by release. See *placement-status upgrade check* for details.

3. Run some commands against the placement API:

   - Install the osc-placement plugin:

     Note: This example uses PyPI and *pip* but if you are using distribution packages you can install the package from their repository. With the move to python3 you will need to specify **pip3** or install **python3-osc-placement** from your distribution.

```
$ pip3 install osc-placement
```

- List available resource classes and traits:

```
$ openstack --os-placement-api-version 1.2 resource class list --
↪sort-column name
+----------------------------+
| name                       |
+----------------------------+
| DISK_GB                    |
| IPV4_ADDRESS               |
| ...                        |

$ openstack --os-placement-api-version 1.6 trait list --sort-column␣
↪name
+----------------------------------------+
| name                                   |
+----------------------------------------+
| COMPUTE_DEVICE_TAGGING                 |
| COMPUTE_NET_ATTACH_INTERFACE           |
| ...                                    |
```

CHAPTER

SEVEN

ADMINISTRATOR GUIDE

7.1 Upgrade

7.1.1 Upgrade Notes

This section provide notes on upgrading to a given target release.

Note: As a reminder, the *placement-status upgrade check* tool can be used to help determine the status of your deployment and how ready it is to perform an upgrade.

For releases prior to Stein, please see the nova upgrade notes.

Train (2.0.0)

The Train release of placement is the first release where placement is available solely from its own project and must be installed separately from nova. If the extracted placement is not already in use, prior to upgrading to Train, the Stein version of placement must be installed. See the next section and *Upgrading from Nova to Placement* for details.

There are no database schema changes in the Train release, but there are checks to confirm that online migrations from Stein have been run. Running *placement-status after upgrading code but prior to restarting the placement service* will notify you of any missing steps and the process to fix it. Once this is done, *placement-manage* should be run to sync the database:

```
$ placement-status upgrade check
+----------------------------------+
| Upgrade Check Results            |
+----------------------------------+
| Check: Missing Root Provider IDs |
| Result: Success                  |
| Details: None                    |
+----------------------------------+
| Check: Incomplete Consumers      |
| Result: Success                  |
| Details: None                    |
+----------------------------------+
$ placement-manage db sync
```

Then the placement service may be restarted.

Stein (1.0.0)

If you are upgrading an existing OpenStack installation from Rocky to Stein, and wish to use the newly extracted placement, you will need to copy some data and configuration settings from nova.

- Configuration and policy files are, by default, located in `/etc/placement`.
- The placement server side settings in `nova.conf` should be moved to a separate placement configuration file `placement.conf`.
- The default configuration value of `[placement]/policy_file` is changed from `placement-policy.yaml` to `policy.yaml`. This config option is changed to *oslo_policy.policy_file* since Train release.
- Several tables in the `nova_api` database need to be migrated to a new `placement` database.

Following these steps will ensure that future changes to placement configuration and code will not conflict with your setup.

As stated above, using the extracted placement code is not required in Stein, there is a copy in the Stein release of Nova. However that code will be deleted in the Train cycle so you must upgrade to external Placement prior to upgrading to Train.

7.1.2 Upgrading from Nova to Placement

This document is for people who are upgrading from an existing Rocky-based installation of OpenStack, where Placement is a part of Nova, to a Stein-based system, using the independently packaged placement service. It is also for people who have already upgraded to Stein but are using the version of the placement service included in Nova in the Stein release.

Upgrading to the extracted placement is not a requirement when upgrading the rest of OpenStack to Stein. The version of the placement service in the Nova Stein release may be used. It is possible to upgrade to Stein and then deploy and switch to the extracted placement at a later time.

The placement code in Nova will be removed in Train so the switch to using extracted placement must happen before upgrading to Train.

Note: The extracted placement code has features and performance and bug fixes that are not present in the placement code in Nova, but no code that is required by Nova. See the release notes for more detail.

If you are installing a new OpenStack, you will want the *installation docs*.

Upgrading to use the extracted placement service requires migrating several database tables from the `nova_api` database to a placement database. Depending on the number of compute hosts in your system and the number of active virtual machines, the amount of data to copy can vary widely. You can get an idea by counting rows in the `resource_providers` and `consumers` tables.

To avoid losing data while performing the copy it is important that writing to the placement database (on either side of the upgrade) is stopped. You may shut down solely the placement service but this will result in errors attempting to use the service from Nova. It is potentially less disruptive to shut down the entire control plane to avoid confusing errors. What strategy is best will vary. This document describes the simple way.

Note: In some installations of nova and placement, data may already be in a database named `placement` and not `nova_api`. If that is the case, you will not need to copy data. Make sure that there are tables and rows in that database and that it is of expected quantity and recently modified (many tables have `created_at` and `updated_at` columns). In some cases the `placement` database will be present *but empty*.

There are database migrations scripts in the placement code repository which may be used to copy the data or as models for your own tooling: mysql-migrate-db.sh and postgresql-migrate-db.sh.

Note: Starting in the Train release, these migration scripts are also packaged with the openstack-placement package on PyPI. Their filenames may be discovered using `pkg_resources` to look in the `placement_db_tools` package:

```
pkg_resources.resource_filename('placement_db_tools', 'mysql-migrate-db.sh')
```

For best results run the database migration on your database host. If you are unable to do this, you will need to take some additional steps below.

This document assumes that the same HTTP endpoint will be used before and after the upgrade. If you need to change that see *Configure User and Endpoints* for guidance.

Initial Steps

1. Install the new placement code on a controller node. This can be openstack-placement from PyPI or you can use packages from a Linux distribution. If you are using the latter be aware that:
   - The name of the package can be found in the *installation docs*.
   - You need to install the packages on a different host from the old nova, to avoid accidentally upgrading before you are ready.
2. Create a `placement` database with appropriate access controls. If you need details on how to do this, see *Create Database*.
3. Create and configure the `placement.conf` file.
   - The default location is `/etc/placement`.
   - Set *`placement_database.connection`* to point to the new database. For example (replacing `PLACEMENT_DBPASS` and `controller` with the appropriate password and host):

     ```
     [placement_database]
     connection = mysql+pymysql://placement:PLACEMENT_DBPASS@controller/
     ↪placement
     ```

   - Configure the *`keystone_authtoken`* section as described in *Install and configure components*.
   - If the following configuration settings are set in the `[placement]` section of `/etc/nova/nova.conf`, move them to a `[placement]` section in `/etc/placement/placement.conf`:
     - *`placement.randomize_allocation_candidates`*

- *placement.incomplete_consumer_project_id*
- *placement.incomplete_consumer_user_id*

4. Move `placement-policy.yaml`, if required.
   - If it exists, move `/etc/nova/placement-policy.yaml` to `/etc/placement/policy.yaml`. If you wish to use a different filename adjust config option `[placement] policy_file`.
5. Configure the database migration tool.
   - Create the configuration file.

   Note: The examples in this guide are using MySQL but if you are using PostgreSQL it is recommended to use the postgresql-migrate-db.sh script since it handles sequences. See bug 2005478 for details.

   ```
   $ mysql-migrate-db.sh --mkconfig /tmp/migrate-db.rc
   ```

   - Edit the file to set the values for the `NOVA_API_USER`, `NOVA_API_PASS`, `PLACEMENT_USER`, and `PLACEMENT_PASS` entries. These are the usernames and passwords for accessing the database.
   - If you are unable to run the migration script on the database host you will need to set `NOVA_API_DB_HOST` and `PLACEMENT_DB_HOST`.
   - Do not change `MIGRATE_TABLES` unless you need to migrate tables incrementally.
6. Configure the web server that will host the placement service. The details of this are beyond the scope of this document. *Install and configure Placement from PyPI* may provide some guidance. **Make sure you also disable the previously running placement service in the web server configuration.**

Migrate the Data

1. Shut down or disable your control plane in whatever way works best for you.
2. Run the migration script:

   ```
   $ mysql-migrate-db.sh --migrate /tmp/migrate-db.rc
   ```

   The `--skip-locks` flag can be used along with `--migrate` in deployments where table locking operations can't be performed. For example, Percona XtraDB Cluster only has experimental support for explicit table locking operations and attempts to use locking will result in errors when PXC Strict Mode is set to ENFORCING.

   If your controller host (the one where you have been editing `/etc/placement/placement.conf`) and database host are not the same, and you have run the migration script on the database host, the final step in the process will fail. This step stamps the database with an initial version (the hash of the first alembic migration) so that future migrations will work properly. From the controller host, you may do it manually with:

   ```
   $ placement-manage db stamp b4ed3a175331
   ```

3. Sync the placement database to be up to date with all migrations:

```
$ placement-manage db sync
```

Finalize the Upgrade

1. Start up the new placement service.
2. Restart your control plane services. If you are upgrading to Stein, continue with the upgrade of the rest of the system.
3. Verify the content of the new service by using the osc-placement tool to list resource providers, allocations and other resources in the service.
4. Verify the integration of placement with the rest of your OpenStack installation by creating and deleting a test server.
5. At some point in the future you may remove the tables in the `nova_api` database that were migrated to the `placement` database.