Nova Documentation

Release 12.0.0.0b2.dev179

OpenStack Foundation

July 14, 2015

1 Compute API References **3**
1.1 REST API Version History 3
1.2 Compute API v2 4

2 Hypervisor Support Matrix **23**
2.1 Hypervisor Support Matrix 23

3 Developer Guide **43**
3.1 Introduction 43
3.2 APIs Development 51
3.3 Concepts 61
3.4 Development policies 83
3.5 Advanced testing and guides 85
3.6 Man Pages 96
3.7 Module Reference 115

4 Indices and tables **701**

Python Module Index **703**

Nova is an OpenStack project designed to provide power massively scalable, on demand, self service access to compute resources.

The developer documentation provided here is continually kept up-to-date based on the latest code, and may not represent the state of the project at any specific prior release.

Note: This is documentation for developers, if you are looking for more general documentation including API, install, operator and user guides see docs.openstack.org

CHAPTER

ONE

COMPUTE API REFERENCES

- v2.1 (CURRENT)
- v2 (SUPPORTED) and v2 extensions (SUPPORTED) (Will be deprecated in the near future.)

API Microversion History:

1.1 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.

1.1.1 2.1

This is the initial version of the v2.1 API which supports microversions. The V2.1 API is from the REST API users's point of view exactly the same as v2.0 except with strong input validation.

A user can specify a header in the API request:

```
X-OpenStack-Nova-API-Version: <version>
```

where `<version>` is any valid api version for this API.

If no version is specified then the API will behave as if a version request of v2.1 was requested.

1.1.2 2.2

Added Keypair type.

A user can request the creation of a certain 'type' of keypair (`ssh` or `x509`) in the `os-keypairs` plugin

If no keypair type is specified, then the default `ssh` type of keypair is created.

Fixes status code for `os-keypairs` create method from 200 to 201

Fixes status code for `os-keypairs` delete method from 202 to 204

1.1.3 2.3

Exposed additional attributes in `os-extended-server-attributes`: `reservation_id`, `launch_index`, `ramdisk_id`, `kernel_id`, `hostname`, `root_device_name`, `userdata`.

Exposed `delete_on_termination` for `attached_volumes` in `os-extended-volumes`.

This change is required for the extraction of EC2 API into a standalone service. It exposes necessary properties absent in public nova APIs yet. Add info for Standalone EC2 API to cut access to Nova DB.

1.1.4 2.4

Show the `reserved` status on a `FixedIP` object in the `os-fixed-ips` API extension. The extension allows one to `reserve` and `unreserve` a fixed IP but the show method does not report the current status.

1.1.5 2.5

Before version 2.5, the command `nova list --ip6 xxx` returns all servers for non-admins, as the filter option is silently discarded. There is no reason to treat ip6 different from ip, though, so we just add this option to the allowed list.

1.1.6 2.6

A new API for getting remote console is added:

```
POST /servers/<uuid>/remote-consoles
{
  "remote_console": {
    "protocol": ["vnc"|"rdp"|"serial"|"spice"],
    "type": ["novnc"|"xpvnc"|"rdp-html5"|"spice-html5"|"serial"]
  }
}
```

Example response:

```
{
  "remote_console": {
    "protocol": "vnc",
    "type": "novnc",
    "url": "http://example.com:6080/vnc_auto.html?token=XYZ"
  }
}
```

The old APIs 'os-getVNCConsole', 'os-getSPICEConsole', 'os-getSerialConsole' and 'os-getRDPConsole' are removed.

Local copy of v2 docs:

1.2 Compute API v2

This section describes the Compute API version 2 and is intended for software developers interested in developing applications using the OpenStack Compute Application Programming Interface (API).

1.2.1 General Compute API v2.0 information

The OpenStack Compute API is defined as a ReSTful HTTP service. The API takes advantage of all aspects of the HTTP protocol (methods, URIs, media types, response codes, etc.) and providers are free to use existing features of the protocol such as caching, persistent connections, and content compression among others. For example, providers who

employ a caching layer may respond with a 203 when a request is served from the cache instead of a 200. Additionally, providers may offer support for conditional **GET** requests using ETags, or they may send a redirect in response to a **GET** request. Clients should be written to account for these differences.

Providers can return information identifying requests in HTTP response headers, for example, to facilitate communication between the provider and client users.

OpenStack Compute is a compute service that provides server capacity in the cloud. Compute Servers come in different flavors of memory, cores, disk space, and CPU, and can be provisioned in minutes. Interactions with Compute Servers can happen programmatically with the OpenStack Compute API.

We welcome feedback, comments, and bug reports at bugs.launchpad.net/nova.

Intended audience

This guide assists software developers who want to develop applications using the OpenStack Compute API. To use this information, you should have access to an account from an OpenStack Compute provider, and you should also be familiar with the following concepts:

- OpenStack Compute service
- ReSTful web services
- HTTP/1.1
- JSON data serialization formats

Concepts

To use the OpenStack Compute API effectively, you should understand several key concepts:

- **Server**

  A virtual machine (VM) instance in the compute system. Flavor and image are requisite elements when creating a server.

- **Flavor**

  An available hardware configuration for a server. Each flavor has a unique combination of disk space, memory capacity and priority for CPU time.

- **Image**

  A collection of files used to create or rebuild a server. Operators provide a number of pre-built OS images by default. You may also create custom images from cloud servers you have launched. These custom images are useful for backup purposes or for producing "gold" server images if you plan to deploy a particular server configuration frequently.

- **Reboot**

  Use this function to perform either a soft or hard reboot of a server. With a soft reboot, the operating system is signaled to restart, which allows for a graceful shutdown of all processes. A hard reboot is the equivalent of power cycling the server. The virtualization platform should ensure that the reboot action has completed successfully even in cases in which the underlying domain/VM is paused or halted/stopped.

- **Rebuild**

  Use this function to remove all data on the server and replaces it with the specified image. Server ID and IP addresses remain the same.

- **Resize**

  Use this function to convert an existing server to a different flavor, in essence, scaling the server up or down. The original server is saved for a period of time to allow rollback if there is a problem. All resizes should be tested and explicitly confirmed, at which time the original server is removed. All resizes are automatically confirmed after 24 hours if you do not confirm or revert them.

- **Pause**

  You can pause a server by making a pause request. This request stores the state of the VM in RAM. A paused instance continues to run in a frozen state.

- **Suspend**

  Administrative users might want to suspend an instance if it is infrequently used or to perform system maintenance. When you suspend an instance, its VM state is stored on disk, all memory is written to disk, and the virtual machine is stopped. Suspending an instance is similar to placing a device in hibernation; memory and vCPUs become available to create other instances.

Reference

For a reference listing for the Compute API v2, see the *Compute API v2 reference (CURRENT)*.

For information about Compute API v 2 extensions, see the *Compute API v2 extensions (CURRENT)*.

1.2.2 Server concepts

For the OpenStack Compute API, a server is a virtual machine (VM) instance in the compute system.

Server status

You can filter the list of servers by image, flavor, name, and status through the respective query parameters.

Servers contain a status attribute that indicates the current server state. You can filter on the server status when you complete a list servers request. The server status is returned in the response body. The server status is one of the following values:

Server status values

- `ACTIVE`: The server is active.
- `BUILD`: The server has not finished the original build process.
- `DELETED`: The server is deleted.
- `ERROR`: The server is in error.
- `HARD_REBOOT`: The server is hard rebooting. This is equivalent to pulling the power plug on a physical server, plugging it back in, and rebooting it.
- `PASSWORD`: The password is being reset on the server.
- `REBOOT`: The server is in a soft reboot state. A reboot command was passed to the operating system.
- `REBUILD`: The server is currently being rebuilt from an image.
- `RESCUE`: The server is in rescue mode.
- `RESIZE`: Server is performing the differential copy of data that changed during its initial copy. Server is down for this stage.

- REVERT_RESIZE: The resize or migration of a server failed for some reason. The destination server is being cleaned up and the original source server is restarting.
- SHUTOFF: The virtual machine (VM) was powered down by the user, but not through the OpenStack Compute API. For example, the user issued a shutdown -h command from within the server instance. If the OpenStack Compute manager detects that the VM was powered down, it transitions the server instance to the SHUTOFF status. If you use the OpenStack Compute API to restart the instance, the instance might be deleted first, depending on the value in the *''shutdown_terminate''* database field on the Instance model.
- SUSPENDED: The server is suspended, either by request or necessity. This status appears for only the following hypervisors: XenServer/XCP, KVM, and ESXi. Administrative users may suspend an instance if it is infrequently used or to perform system maintenance. When you suspend an instance, its VM state is stored on disk, all memory is written to disk, and the virtual machine is stopped. Suspending an instance is similar to placing a device in hibernation; memory and vCPUs become available to create other instances.
- UNKNOWN: The state of the server is unknown. Contact your cloud provider.
- VERIFY_RESIZE: System is awaiting confirmation that the server is operational after a move or resize.

The compute provisioning algorithm has an anti-affinity property that attempts to spread customer VMs across hosts. Under certain situations, VMs from the same customer might be placed on the same host. hostId represents the host your server runs on and can be used to determine this scenario if it is relevant to your application.

Note: HostId is unique *per account* and is not globally unique.

Server creation

Status Transition:

BUILD

ACTIVE

BUILD

ERROR (on error)

When you create a server, the operation asynchronously provisions a new server. The progress of this operation depends on several factors including location of the requested image, network I/O, host load, and the selected flavor. The progress of the request can be checked by performing a **GET** on /servers/*''id''*, which returns a progress attribute (from 0% to 100% complete). The full URL to the newly created server is returned through the Location header and is available as a self and bookmark link in the server representation. Note that when creating a server, only the server ID, its links, and the administrative password are guaranteed to be returned in the request. You can retrieve additional attributes by performing subsequent **GET** operations on the server.

Server passwords

You can specify a password when you create the server through the optional adminPass attribute. The specified password must meet the complexity requirements set by your OpenStack Compute provider. The server might enter an ERROR state if the complexity requirements are not met. In this case, a client can issue a change password action to reset the server password.

If a password is not specified, a randomly generated password is assigned and returned in the response object. This password is guaranteed to meet the security requirements set by the compute provider. For security reasons, the password is not returned in subsequent **GET** calls.

Server metadata

Custom server metadata can also be supplied at launch time. The maximum size of the metadata key and value is 255 bytes each. The maximum number of key-value pairs that can be supplied per server is determined by the compute provider and may be queried via the maxServerMeta absolute limit.

Server networks

Networks to which the server connects can also be supplied at launch time. One or more networks can be specified. User can also specify a specific port on the network or the fixed IP address to assign to the server interface.

Server personality

You can customize the personality of a server instance by injecting data into its file system. For example, you might want to insert ssh keys, set configuration files, or store data that you want to retrieve from inside the instance. This feature provides a minimal amount of launch-time personalization. If you require significant customization, create a custom image.

Follow these guidelines when you inject files:

- The maximum size of the file path data is 255 bytes.
- Encode the file contents as a Base64 string. The maximum size of the file contents is determined by the compute provider and may vary based on the image that is used to create the server

Considerations

The maximum limit refers to the number of bytes in the decoded data and not the number of characters in the encoded data.

- You can inject text files only. You cannot inject binary or zip files into a new build.
- The maximum number of file path/content pairs that you can supply is also determined by the compute provider and is defined by the maxPersonality absolute limit.
- The absolute limit, maxPersonalitySize, is a byte limit that is guaranteed to apply to all images in the deployment. Providers can set additional per-image personality limits.

The file injection might not occur until after the server is built and booted.

During file injection, any existing files that match specified files are renamed to include the BAK extension appended with a time stamp. For example, if the `/etc/passwd` file exists, it is backed up as `/etc/passwd.bak.1246036261.5785`.

After file injection, personality files are accessible by only system administrators. For example, on Linux, all files have root and the root group as the owner and group owner, respectively, and allow user and group read access only (octal 440).

Server access addresses

In a hybrid environment, the IP address of a server might not be controlled by the underlying implementation. Instead, the access IP address might be part of the dedicated hardware; for example, a router/NAT device. In this case, the addresses provided by the implementation cannot actually be used to access the server (from outside the local LAN). Here, a separate *access address* may be assigned at creation time to provide access to the server. This address may not be directly bound to a network interface on the server and may not necessarily appear when a server's addresses are

queried. Nonetheless, clients that must access the server directly are encouraged to do so via an access address. In the example below, an IPv4 address is assigned at creation time.

Example: Create server with access IP: JSON request

```
{
   "server":{
      "name":"new-server-test",
      "imageRef":"52415800-8b69-11e0-9b19-734f6f006e54",
      "flavorRef":"52415800-8b69-11e0-9b19-734f1195ff37",
      "accessIPv4":"67.23.10.132"
   }
}
```

Note: Both IPv4 and IPv6 addresses may be used as access addresses and both addresses may be assigned simultaneously as illustrated below. Access addresses may be updated after a server has been created.

Example: Create server with multiple access IPs: JSON request

```
{
   "server":{
      "name":"new-server-test",
      "imageRef":"52415800-8b69-11e0-9b19-734f6f006e54",
      "flavorRef":"52415800-8b69-11e0-9b19-734f1195ff37",
      "accessIPv4":"67.23.10.132",
      "accessIPv6":"::babe:67.23.10.132"
   }
}
```

1.2.3 Authentication

Each HTTP request against the OpenStack Compute system requires the inclusion of specific authentication credentials. A single deployment may support multiple authentication schemes (OAuth, Basic Auth, Token). The authentication scheme is provided by the OpenStack Identity service. You can contact your provider to determine the best way to authenticate against the Compute API.

Note: Some authentication schemes may require that the API operate using SSL over HTTP (HTTPS).

1.2.4 Extensions

The OpenStack Compute API v2.0 is extensible. Extensions serve two purposes: They allow the introduction of new features in the API without requiring a version change and they allow the introduction of vendor specific niche functionality. Applications can programmatically list available extensions by performing a **GET** on the `/extensions` URI. Note that this is a versioned request; that is, an extension available in one API version might not be available in another.

Extensions may also be queried individually by their unique alias. This provides the simplest method of checking if an extension is available because an unavailable extension issues an itemNotFound (404) response.

Extensions may define new data types, parameters, actions, headers, states, and resources. In XML, additional elements and attributes can be defined. These elements must be defined in the namespace for the extension. In JSON, the alias must be used. The volumes element is defined in the `RS-CBS` namespace. Extended headers are always prefixed with `X-` followed by the alias and a dash: (`X-RS-CBS-HEADER1`). States and parameters must be prefixed with the extension alias followed by a colon. For example, an image might be in the `RS-PIE:PrepareShare` state.

Important

Applications should ignore response data that contains extension elements. An extended state should always be treated as an UNKNOWN state if the application does not support the extension. Applications should also verify that an extension is available before submitting an extended request.

Example: Extended server: JSON response


```
{
    "servers": [
        {
            "id": "52415800-8b69-11e0-9b19-734f6af67565",
            "tenant_id": "1234",
            "user_id": "5678",
            "name": "sample-server",
            "updated": "2010-10-10T12:00:00Z",
            "created": "2010-08-10T12:00:00Z",
            "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
            "status": "BUILD",
            "progress": 60,
            "accessIPv4" : "67.23.10.132",
            "accessIPv6" : "::babe:67.23.10.132",
            "image" : {
                "id": "52415800-8b69-11e0-9b19-734f6f006e54",
                "links": [
                    {
                        "rel": "self",
                        "href": "http://servers.api.openstack.org/v2/1234/images/52415800-8b69-11e0-9
                    },
                    {
                        "rel": "bookmark",
                        "href": "http://servers.api.openstack.org/1234/images/52415800-8b69-11e0-9b19
                    }
                ]
            },
            "flavor" : {
                "id": "52415800-8b69-11e0-9b19-734f216543fd",
                "links": [
                    {
                        "rel": "self",
                        "href": "http://servers.api.openstack.org/v2/1234/flavors/52415800-8b69-11e0-
                    },
                    {
                        "rel": "bookmark",
                        "href": "http://servers.api.openstack.org/1234/flavors/52415800-8b69-11e0-9b1
                    }
                ]
            },
            "addresses": {
                "public" : [
                    {
                        "version": 4,
                        "addr": "67.23.10.132"
                    },
                    {
                        "version": 6,
                        "addr": "::babe:67.23.10.132"
                    },
                    {
```

```
                        "version": 4,
                        "addr": "67.23.10.131"
                    },
                    {
                        "version": 6,
                        "addr": "::babe:4317:0A83"
                    }
                ],
                "private" : [
                    {
                        "version": 4,
                        "addr": "10.176.42.16"
                    },
                    {
                        "version": 6,
                        "addr": "::babe:10.176.42.16"
                    }
                ]
            },
            "metadata": {
                "Server Label": "Web Head 1",
                "Image Version": "2.1"
            },
            "links": [
                {
                    "rel": "self",
                    "href": "http://servers.api.openstack.org/v2/1234/servers/52415800-8b69-11e0-9b1
                },
                {
                    "rel": "bookmark",
                    "href": "http://servers.api.openstack.org/1234/servers/52415800-8b69-11e0-9b19-7
                }
            ],
            "RS-CBS:volumes": [
                {
                    "name": "OS",
                    "href": "https://cbs.api.rackspacecloud.com/12934/volumes/19"
                },
                {
                    "name": "Work",
                    "href": "https://cbs.api.rackspacecloud.com/12934/volumes/23"
                }
            ]
        }
    ]
}
```

Example: Extended action: JSON response

```
{
   "RS-CBS:attach-volume":{
      "href":"https://cbs.api.rackspacecloud.com/12934/volumes/19"
   }
}
```

1.2.5 Faults

Synchronous faults

When an error occurs at request time, the system also returns additional information about the fault in the body of the response.

Example: Fault: JSON response

```
{
   "computeFault":{
      "code":500,
      "message":"Fault!",
      "details":"Error Details..."
   }
}
```

The error code is returned in the body of the response for convenience. The message section returns a human-readable message that is appropriate for display to the end user. The details section is optional and may contain information—for example, a stack trace—to assist in tracking down an error. The detail section might or might not be appropriate for display to an end user.

The root element of the fault (such as, computeFault) might change depending on the type of error. The following is a list of possible elements along with their associated error codes.

Fault elements and error codes

- `computeFault`: 500, 400, other codes possible
- `notImplemented`: 501
- `serverCapacityUnavailable`: 503
- `serviceUnavailable`: 503
- `badRequest`: 400
- `unauthorized`: 401
- `forbidden`: 403
- `resizeNotAllowed`: 403
- `itemNotFound`: 404
- `badMethod`: 405
- `backupOrResizeInProgress`: 409
- `buildInProgress`: 409
- `conflictingRequest`: 409
- `overLimit`: 413
- `badMediaType`: 415

Example: Item Not Found fault: JSON response

```
{
   "itemNotFound":{
      "code":404,
      "message":"Not Found",
```

```
      "details":"Error Details..."
   }
}
```

From an XML schema perspective, all API faults are extensions of the base ComputeAPIFault fault type. When working with a system that binds XML to actual classes (such as JAXB), you should use ComputeAPIFault as a catch-all if you do not want to distinguish between individual fault types.

The OverLimit fault is generated when a rate limit threshold is exceeded. For convenience, the fault adds a retryAfter attribute that contains the content of the Retry-After header in XML Schema 1.0 date/time format.

Example: Over Limit fault: JSON response

```
{
    "overLimit" : {
        "code" : 413,
        "message" : "OverLimit Retry...",
        "details" : "Error Details...",
        "retryAfter" : "2010-08-01T00:00:00Z"
    }
}
```

Asynchronous faults

An error may occur in the background while a server or image is being built or while a server is executing an action. In these cases, the server or image is placed in an `ERROR` state and the fault is embedded in the offending server or image. Note that these asynchronous faults follow the same format as the synchronous ones. The fault contains an error code, a human readable message, and optional details about the error. Additionally, asynchronous faults may also contain a created timestamp that specify when the fault occurred.

Example: Server in error state: JSON response

```
{
    "server": {
        "id": "52415800-8b69-11e0-9b19-734f0000ffff",
        "tenant_id": "1234",
        "user_id": "5678",
        "name": "sample-server",
        "created": "2010-08-10T12:00:00Z",
        "hostId": "e4d909c290d0fb1ca068ffafff22cbd0",
        "status": "ERROR",
        "progress": 66,
        "image" : {
            "id": "52415800-8b69-11e0-9b19-734f6f007777"
        },
        "flavor" : {
            "id": "52415800-8b69-11e0-9b19-734f216543fd"
        },
        "fault" : {
            "code" : 404,
            "created": "2010-08-10T11:59:59Z",
            "message" : "Could not find image 52415800-8b69-11e0-9b19-734f6f007777",
            "details" : "Fault details"
        },
        "links": [
            {
                "rel": "self",
                "href": "http://servers.api.openstack.org/v2/1234/servers/52415800-8b69-11e0-9b19-734
```

```
            },
            {
                "rel": "bookmark",
                "href": "http://servers.api.openstack.org/1234/servers/52415800-8b69-11e0-9b19-734f0
            }
        ]
    }
}
```

Example: Image in error state: JSON response

```
{
    "image" : {
        "id" : "52415800-8b69-11e0-9b19-734f5736d2a2",
        "name" : "My Server Backup",
        "created" : "2010-08-10T12:00:00Z",
        "status" : "SAVING",
        "progress" : 89,
        "server" : {
            "id": "52415800-8b69-11e0-9b19-734f335aa7b3"
        },
        "fault" : {
            "code" : 500,
            "message" : "An internal error occurred",
            "details" : "Error details"
        },
        "links": [
            {
                "rel" : "self",
                "href" : "http://servers.api.openstack.org/v2/1234/images/52415800-8b69-11e0-9b19-734
            },
            {
                "rel" : "bookmark",
                "href" : "http://servers.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f5
            }
        ]
    }
}
```

1.2.6 Limits

Accounts may be pre-configured with a set of thresholds (or limits) to manage capacity and prevent abuse of the system. The system recognizes two kinds of limits: *rate limits* and *absolute limits*. Rate limits are thresholds that are reset after a certain amount of time passes. Absolute limits are fixed. Limits are configured by operators and may differ from one deployment of the OpenStack Compute service to another. Please contact your provider to determine the limits that apply to your account. Your provider may be able to adjust your account's limits if they are too low. Also see the API Reference for *Limits*.

Rate limits

Rate limits are specified in terms of both a human-readable wild-card URI and a machine-processable regular expression. The human-readable limit is intended for displaying in graphical user interfaces. The machine-processable form is intended to be used directly by client applications.

The regular expression boundary matcher "^" for the rate limit takes effect after the root URI path. For example, the regular expression ^/servers would match the bolded portion of the following URI:

https://servers.api.openstack.org/v2/3542812**/servers**.

Table: Sample rate limits

Verb	URI	RegEx	Default
POST	*	.*	120/min
POST	*/servers	^/servers	120/min
PUT	*	.*	120/min
GET	*changes-since*	.*changes-since.*	120/min
DELETE	*	.*	120/min
GET	*/os-fping*	^/os-fping	12/min

Rate limits are applied in order relative to the verb, going from least to most specific.

In the event a request exceeds the thresholds established for your account, a 413 HTTP response is returned with a `Retry-After` header to notify the client when they can attempt to try again.

Absolute limits

Absolute limits are specified as name/value pairs. The name of the absolute limit uniquely identifies the limit within a deployment. Please consult your provider for an exhaustive list of absolute value names. An absolute limit value is always specified as an integer. The name of the absolute limit determines the unit type of the integer value. For example, the name maxServerMeta implies that the value is in terms of server metadata items.

Table: Sample absolute limits

Name	Value	Description
maxTotalRAMSize	51200	Maximum total amount of RAM (MB)
maxServerMeta	5	Maximum number of metadata items associated with a server.
maxImageMeta	5	Maximum number of metadata items associated with an image.
maxPersonality	5	The maximum number of file path/content pairs that can be supplied on server build.
maxPersonality-Size	10240	The maximum size, in bytes, for each personality file.

Determine limits programmatically

Applications can programmatically determine current account limits. For information, see *Limits*.

1.2.7 Links and references

Often resources need to refer to other resources. For example, when creating a server, you must specify the image from which to build the server. You can specify the image by providing an ID or a URL to a remote image. When providing an ID, it is assumed that the resource exists in the current OpenStack deployment.

Example: ID image reference: JSON request

```
{
   "server":{
      "flavorRef":"http://openstack.example.com/openstack/flavors/1",
      "imageRef":"http://openstack.example.com/openstack/images/70a599e0-31e7-49b7-b260-868f441e862b
      "metadata":{
         "My Server Name":"Apache1"
      },
      "name":"new-server-test",
      "personality":[
```

```
            {
                "contents":"ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBpdCBtb3ZlcyBpbiBqdXN0IHN1Y2ggY
                "path":"/etc/banner.txt"
            }
        ]
    }
}
```

Example: Full image reference: JSON request

```
{
    "server": {
        "name": "server-test-1",
        "imageRef": "b5660a6e-4b46-4be3-9707-6b47221b454f",
        "flavorRef": "2",
        "max_count": 1,
        "min_count": 1,
        "networks": [
            {
                "uuid": "d32019d3-bc6e-4319-9c1d-6722fc136a22"
            }
        ],
        "security_groups": [
            {
                "name": "default"
            },
            {
                "name": "another-secgroup-name"
            }
        ]
    }
}
```

For convenience, resources contain links to themselves. This allows a client to easily obtain rather than construct resource URIs. The following types of link relations are associated with resources:

- A `self` link contains a versioned link to the resource. Use these links when the link is followed immediately.
- A `bookmark` link provides a permanent link to a resource that is appropriate for long term storage.
- An `alternate` link can contain an alternate representation of the resource. For example, an OpenStack Compute image might have an alternate representation in the OpenStack Image service.

Note: The type attribute provides a hint as to the type of representation to expect when following the link.

Example: Server with self links: JSON

```
{
   "server":{
      "id":"52415800-8b69-11e0-9b19-734fcece0043",
      "name":"my-server",
      "links":[
         {
            "rel":"self",
            "href":"http://servers.api.openstack.org/v2/1234/servers/52415800-8b69-11e0-9b19-734fcece
         },
         {
            "rel":"bookmark",
            "href":"http://servers.api.openstack.org/1234/servers/52415800-8b69-11e0-9b19-734fcece004
```

```
            }
        ]
    }
}
```

Example: Server with alternate link: JSON

```
{
    "image" : {
        "id" : "52415800-8b69-11e0-9b19-734f5736d2a2",
        "name" : "My Server Backup",
        "links": [
            {
                "rel" : "self",
                "href" : "http://servers.api.openstack.org/v2/1234/images/52415800-8b69-11e0-9b19-734
            },
            {
                "rel" : "bookmark",
                "href" : "http://servers.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f5
            },
            {
                "rel"  : "alternate",
                "type" : "application/vnd.openstack.image",
                "href" : "http://glance.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f573
            }
        ]
    }
}
```

1.2.8 Paginated collections

To reduce load on the service, list operations return a maximum number of items at a time. The maximum number of items returned is determined by the compute provider. To navigate the collection, the *''limit''* and *''marker''* parameters can be set in the URI. For example:

```
?limit=100&marker=1234
```

The *''marker''* parameter is the ID of the last item in the previous list. By default, the service sorts items by create time in descending order. When the service cannot identify a a create time, it sorts items by ID. The *''limit''* parameter sets the page size. Both parameters are optional. If the client requests a *''limit''* beyond that which is supported by the deployment an overLimit (413) fault may be thrown. A marker with an invalid ID returns a badRequest (400) fault.

For convenience, collections should contain atom `next` links. They may optionally also contain `previous` links but the current implementation does not contain `previous` links. The last page in the list does not contain a "next" link. The following examples illustrate three pages in a collection of images. The first page was retrieved through a **GET** to `http://servers.api.openstack.org/v2/1234/servers?limit=1`. In these examples, the *''limit''* parameter sets the page size to a single item. Subsequent links honor the initial page size. Thus, a client can follow links to traverse a paginated collection without having to input the *''marker''* parameter.

Example: Servers collection: JSON (first page)

```
{
   "servers_links":[
      {
         "href":"https://servers.api.openstack.org/v2/1234/servers?limit=1&marker=fc45ace4-3398-447b-
         "rel":"next"
      }
```

```
    ],
    "servers":[
        {
            "id":"fc55acf4-3398-447b-8ef9-72a42086d775",
            "links":[
                {
                    "href":"https://servers.api.openstack.org/v2/1234/servers/fc45ace4-3398-447b-8ef9-72a2
                    "rel":"self"
                },
                {
                    "href":"https://servers.api.openstack.org/v2/1234/servers/fc45ace4-3398-447b-8ef9-72a2
                    "rel":"bookmark"
                }
            ],
            "name":"elasticsearch-0"
        }
    ]
}
```


In JSON, members in a paginated collection are stored in a JSON array named after the collection. A JSON object may also be used to hold members in cases where using an associative array is more practical. Properties about the collection itself, including links, are contained in an array with the name of the entity an underscore (_) and `links`. The combination of the objects and arrays that start with the name of the collection and an underscore represent the collection in JSON. The approach allows for extensibility of paginated collections by allowing them to be associated with arbitrary properties. It also allows collections to be embedded in other objects as illustrated below. Here, a subset of metadata items are presented within the image. Clients must follow the "next" link to retrieve the full set of metadata.

Example: Paginated metadata: JSON


```
{
    "server": {
        "id": "52415800-8b69-11e0-9b19-734f6f006e54",
        "name": "Elastic",
        "metadata": {
            "Version": "1.3",
            "ServiceType": "Bronze"
        },
        "metadata_links": [
            {
                "rel": "next",
                "href": "https://servers.api.openstack.org/v2/1234/servers/fc55acf4-3398-447b-8ef9-72
            }
        ],
        "links": [
            {
                "rel": "self",
                "href": "https://servers.api.openstack.org/v2/1234/servers/fc55acf4-3398-447b-8ef9-72
            }
        ]
    }
}
```


1.2.9 Efficient polling with the Changes-Since parameter

The ReST API allows you to poll for the status of certain operations by performing a **GET** on various elements. Rather than re-downloading and re-parsing the full status at each polling interval, your ReST client may use the

''changes-since'' parameter to check for changes since a previous request. The *''changes-since''* time is specified as an ISO 8601 dateTime (2011-01-24T17:08Z). The form for the timestamp is CCYY-MM-DDThh:mm:ss. An optional time zone may be written in by appending the form ±hh:mm which describes the timezone as an offset from UTC. When the timezone is not specified (2011-01-24T17:08), the UTC timezone is assumed. If nothing has changed since the *''changes-since''* time, an empty list is returned. If data has changed, only the items changed since the specified time are returned in the response. For example, performing a **GET** against https://api.servers.openstack.org/v2/224532/servers?*''changes-since''*=2015-01-24T17:08Z would list all servers that have changed since Mon, 24 Jan 2015 17:08:00 UTC.

To allow clients to keep track of changes, the changes-since filter displays items that have been *recently* deleted. Both images and servers contain a `DELETED` status that indicates that the resource has been removed. Implementations are not required to keep track of deleted resources indefinitely, so sending a changes since time in the distant past may miss deletions.

1.2.10 Request and response formats

The OpenStack Compute API supports JSON request and response formats.

Request format

Use the `Content-Type` request header to specify the request format. This header is required for operations that have a request body.

The syntax for the `Content-Type` header is:

```
Content-Type: application/FORMAT
```

Where `FORMAT` is `json`.

Response format

Use one of the following methods to specify the response format:

`Accept` header The syntax for the `Accept` header is:

```
Accept: application/FORMAT
```

Where *''FORMAT''* is `json` and the default format is `json`.

Query extension Add a `.json` extension to the request URI. For example, the `.json` extension in the following list servers URI request specifies that the response body is to be returned in JSON format:

GET *''publicURL''*/servers.json

If you do not specify a response format, JSON is the default.

Request and response examples

You can serialize a response in a different format from the request format.

The examples below show a request body in JSON format.

Note: Though you may find outdated documents with XML examples, XML support in requests and responses has been deprecated for the Compute API v2 (stable) and v2.1 (experimental).

Example: JSON request with headers

POST /v2/010101/servers HTTP/1.1
Host: servers.api.openstack.org
Content-Type: application/json
Accept: application/xml
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb

```
{
    "server": {
        "name": "server-test-1",
        "imageRef": "b5660a6e-4b46-4be3-9707-6b47221b454f",
        "flavorRef": "2",
        "max_count": 1,
        "min_count": 1,
        "networks": [
            {
                "uuid": "d32019d3-bc6e-4319-9c1d-6722fc136a22"
            }
        ],
        "security_groups": [
            {
                "name": "default"
            },
            {
                "name": "another-secgroup-name"
            }
        ]
    }
}
```

1.2.11 Versions

The OpenStack Compute API uses both a URI and a MIME type versioning scheme. In the URI scheme, the first element of the path contains the target version identifier (e.g. https://servers.api.openstack.org/ v2.0/...). The MIME type versioning scheme uses HTTP content negotiation where the `Accept` or `Content-Type` headers contains a MIME type that identifies the version (application/vnd.openstack.compute.v2+json). A version MIME type is always linked to a base MIME type, such as application/json. If conflicting versions are specified using both an HTTP header and a URI, the URI takes precedence.

Example: Request with MIME type versioning

```
GET /214412/images HTTP/1.1
Host: servers.api.openstack.org
Accept: application/vnd.openstack.compute.v2+json
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

Example: Request with URI versioning

```
GET /v2/214412/images HTTP/1.1
Host: servers.api.openstack.org
Accept: application/json
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

Permanent Links

The MIME type versioning approach allows for the creating of permanent links, because the version scheme is not specified in the URI path: https://api.servers.openstack.org/224532/servers/123.

If a request is made without a version specified in the URI or via HTTP headers, then a multiple-choices response (300) follows that provides links and MIME types to available versions.

Example: Multiple choices: JSON response


```
{
   "choices":[
      {
         "id":"v1.0",
         "status":"DEPRECATED",
         "links":[
            {
               "rel":"self",
               "href":"http://servers.api.openstack.org/v1.0/1234/servers/52415800-8b69-11e0-9b19-734
            }
         ],
         "media-types":[
            {
               "base":"application/json",
               "type":"application/vnd.openstack.compute.v1.0+json"
            }
         ]
      },
      {
         "id":"v2",
         "status":"CURRENT",
         "links":[
            {
               "rel":"self",
               "href":"http://servers.api.openstack.org/v2/1234/servers/52415800-8b69-11e0-9b19-734f
            }
         ],
         "media-types":[
            {
               "base":"application/json",
               "type":"application/vnd.openstack.compute.v2+json"
            }
         ]
      }
   ]
}
```


New features and functionality that do not break API-compatibility are introduced in the current version of the API as extensions and the URI and MIME types remain unchanged. Features or functionality changes that would necessitate a break in API-compatibility require a new version, which results in URI and MIME type version being updated accordingly. When new API versions are released, older versions are marked as `DEPRECATED`. Providers should work with developers and partners to ensure there is adequate time to migrate to the new version before deprecated versions are discontinued.

Your application can programmatically determine available API versions by performing a **GET** on the root URL (i.e. with the version and everything to the right of it truncated) returned from the authentication system.

You can also obtain additional information about a specific version by performing a **GET** on the base version URL (such as, `https://servers.api.openstack.org/v2/`). Version request URLs must always end with a

trailing slash (/). If you omit the slash, the server might respond with a 302 redirection request. Format extensions can be placed after the slash (such as, `https://servers.api.openstack.org/v2/.json`).

Note: This special case does not hold true for other API requests. In general, requests such as `/servers.json` and `/servers/.json` are handled equivalently.

For examples of the list versions and get version details requests and responses, see *API versions*.

The detailed version response contains pointers to both a human-readable and a machine-processable description of the API service. The machine-processable description is written in the Web Application Description Language (WADL).

CHAPTER

TWO

HYPERVISOR SUPPORT MATRIX

2.1 Hypervisor Support Matrix

When considering which capabilities should be marked as mandatory the following general guiding principles were applied

- **Inclusivity** - people have shown ability to make effective use of a wide range of virtualization technologies with broadly varying featuresets. Aiming to keep the requirements as inclusive as possible, avoids second-guessing what a user may wish to use the cloud compute service for.
- **Bootstrapping** - a practical use case test is to consider that starting point for the compute deploy is an empty data center with new machines and network connectivity. The look at what are the minimum features required of a compute service, in order to get user instances running and processing work over the network.
- **Competition** - an early leader in the cloud compute service space was Amazon EC2. A sanity check for whether a feature should be mandatory is to consider whether it was available in the first public release of EC2. This had quite a narrow featureset, but none the less found very high usage in many use cases. So it serves to illustrate that many features need not be considered mandatory in order to get useful work done.
- **Reality** - there are many virt drivers currently shipped with Nova, each with their own supported feature set. Any feature which is missing in at least one virt driver that is already in-tree, must by inference be considered optional until all in-tree drivers support it. This does not rule out the possibility of a currently optional feature becoming mandatory at a later date, based on other principles above.

Summary

Feature	*Status*	**Hyper-V**	**Ironic**	**Libvirt KVM (ppc64)**	**Libvirt KVM (s390x)**	**Lib**
Attach block volume to instance	optional	✓	✖	✓	✓	✓
Detach block volume from instance	optional	✓	✖	✓	✓	✓
Set the host in a maintenance mode	optional	✖	✖	✖	✖	✖
Guest instance status	mandatory	✓	✓	✓	✓	✓
Guest host status	optional	✓	✖	✓	✓	✓
Live migrate instance across hosts	optional	✓	✖	✓	✓	✓
Launch instance	mandatory	✓	✓	✓	✓	✓
Stop instance CPUs	optional	✓	✖	✓	✓	✓
Reboot instance	optional	✓	✓	✓	✓	✓
Rescue instance	optional	✖	✖	✓	✓	✓
Resize instance	optional	✓	✓	✓	✓	✓
Restore instance	optional	✓	✖	✓	✓	✓
Service control	optional	✖	✖	✓	✓	✓
Set instance admin password	optional	✖	✖	✖	✖	✖
Save snapshot of instance disk	optional	✓	✖	✓	✓	✓
Suspend instance	optional	✓	✖	✓	✓	✓

Table 2

Feature	*Status*	Hyper-V	Ironic	Libvirt KVM (ppc64)	Libvirt KVM (s390x)	Lib
Swap block volumes	optional	✖	✖	✓	✓	✓
Shutdown instance	mandatory	✓	✓	✓	✓	✓
Resume instance CPUs	optional	✓	✖	✓	✓	✓
Auto configure disk	optional	✓	✖	✖	✖	✖
Instance disk I/O limits	optional	✖	✖	✓	✓	✓
Config drive support	choice	✓	✖	✖	✓	✓
Inject files into disk image	optional	✖	✖	✖	✖	✓
Inject guest networking config	optional	✖	✖	✖	✖	✓
Remote desktop over RDP	choice	✓	✖	✖	✖	✖
View serial console logs	choice	✓	✖	✖	✓	✓
Remote interactive serial console	choice	✖	?	?	✓	✓
Remote desktop over SPICE	choice	✖	✖	✖	✖	✓
Remote desktop over VNC	choice	✖	✖	✖	✖	✓
Block storage support	optional	✓	✖	✓	✓	✓
Block storage over fibre channel	optional	✖	✖	✖	✓	✓
Block storage over iSCSI	condition	✓	✖	✓	✓	✓
CHAP authentication for iSCSI	optional	✓	✖	✓	✓	✓
Image storage support	mandatory	✓	✓	✓	✓	✓
Network firewall rules	optional	✖	✖	✓	✓	✓
Network routing	optional	✖	✓	✖	✓	✓
Network security groups	optional	✖	✖	✓	✓	✓
Flat networking	choice	✓	✓	✓	✓	✓
VLAN networking	choice	✖	✖	✓	✓	✓

Details

- **Attach block volume to instance Status: optional.** The attach volume operation provides a means to hotplug additional block storage to a running instance. This allows storage capabilities to be expanded without interruption of service. In a cloud model it would be more typical to just spin up a new instance with large storage, so the ability to hotplug extra storage is for those cases where the instance is considered to be more of a pet than cattle. Therefore this operation is not considered to be mandatory to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Detach block volume from instance Status: optional.** See notes for attach volume operation.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Set the host in a maintenance mode Status: optional.** This operation allows a host to be placed into maintenance mode, automatically triggering migration of any running instances to an alternative host and preventing new instances from being launched. This is not considered to be a mandatory operation to support. The CLI command is "nova host-update <host>". The driver methods to implement are "host_maintenance_mode" and "set_host_enabled".

  drivers:

  - **Libvirt KVM (s390x):** `missing`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `missing`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `missing`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Guest instance status Status: mandatory.** Provides a quick report on information about the guest instance, including the power state, memory allocation, CPU allocation, number of vCPUs and cummulative CPU execution time. As well as being informational, the power state is used by the compute manager for tracking changes in guests. Therefore this operation is considered mandatory to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`

  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `complete`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Guest host status Status: optional.** Unclear what this refers to

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Live migrate instance across hosts Status: optional.** Live migration provides a way to move an instance off one compute host, to another compute host. Administrators may use this to evacuate instances from a host that needs to undergo maintenance tasks, though of course this may not help if the host is already suffering a failure. In general instances are considered cattle rather than pets, so it is expected that an instance is liable to be killed if host maintenance is required. It is technically challenging for some hypervisors to provide support for the live migration operation, particularly those built on the container based virtualization. Therefore this operation is not considered mandatory to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`

  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing` https://bugs.launchpad.net/nova/+bug/1192192
  - **Libvirt Parallels CT:** `missing`

- **Launch instance Status: mandatory.** Importing pre-existing running virtual machines on a host is considered out of scope of the cloud paradigm. Therefore this operation is mandatory to support in drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `complete`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Stop instance CPUs Status: optional.** Stopping an instances CPUs can be thought of as roughly equivalent to suspend-to-RAM. The instance is still present in memory, but execution has stopped. The problem, however, is that there is no mechanism to inform the guest OS that this takes place, so upon unpausing, its clocks will no longer report correct time. For this reason hypervisor vendors generally discourage use of this feature and some do not even implement it. Therefore this operation is considered optional to support in drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`

  - **VMware vCenter:** missing
  - **Libvirt Parallels CT:** missing

- **Reboot instance Status: optional.** It is reasonable for a guest OS administrator to trigger a graceful reboot from inside the instance. A host initiated graceful reboot requires guest co-operation and a non-graceful reboot can be achieved by a combination of stop+start. Therefore this operation is considered optional.

  drivers:

  - **Libvirt KVM (s390x):** complete
  - **Libvirt KVM (ppc64):** complete
  - **Libvirt KVM (x86):** complete
  - **Libvirt LXC:** complete
  - **Hyper-V:** complete
  - **Libvirt Parallels VM:** complete
  - **Libvirt QEMU (x86):** complete
  - **XenServer:** complete
  - **Libvirt Xen:** complete
  - **Ironic:** complete
  - **VMware vCenter:** complete
  - **Libvirt Parallels CT:** complete

- **Rescue instance Status: optional.** The rescue operation starts an instance in a special configuration whereby it is booted from an special root disk image. The goal is to allow an administrator to recover the state of a broken virtual machine. In general the cloud model considers instances to be cattle, so if an instance breaks the general expectation is that it be thrown away and a new instance created. Therefore this operation is considered optional to support in drivers.

  drivers:

  - **Libvirt KVM (s390x):** complete
  - **Libvirt KVM (ppc64):** complete
  - **Libvirt KVM (x86):** complete
  - **Libvirt LXC:** missing
  - **Hyper-V:** missing
  - **Libvirt Parallels VM:** missing
  - **Libvirt QEMU (x86):** complete
  - **XenServer:** complete
  - **Libvirt Xen:** complete
  - **Ironic:** missing
  - **VMware vCenter:** complete
  - **Libvirt Parallels CT:** missing

- **Resize instance Status: optional.** The resize operation allows the user to change a running instance to match the size of a different flavor from the one it was initially launched with. There are many different flavor attributes that potentially need to be updated. In general it is technically challenging for a hypervisor to support the alteration of all relevant config settings for a running instance. Therefore this operation is considered optional to support in drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `partial` Only certain ironic drivers support this
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Restore instance Status: optional.** See notes for the suspend operation

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Service control Status: optional.** Something something, dark side, something something. Hard to claim this is mandatory when no one seems to know what "Service control" refers to in the context of virt drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`

  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Set instance admin password Status: optional.** Provides a mechanism to re(set) the password of the administrator account inside the instance operating system. This requires that the hypervisor has a way to communicate with the running guest operating system. Given the wide range of operating systems in existence it is unreasonable to expect this to be practical in the general case. The configdrive and metadata service both provide a mechanism for setting the administrator password at initial boot time. In the case where this operation were not available, the administrator would simply have to login to the guest and change the password in the normal manner, so this is just a convenient optimization. Therefore this operation is not considered mandatory for drivers to support.

  drivers:

  - **Libvirt KVM (s390x):** `missing`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `missing`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `missing`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Save snapshot of instance disk Status: optional.** The snapshot operation allows the current state of the instance root disk to be saved and uploaded back into the glance image repository. The instance can later be booted again using this saved image. This is in effect making the ephemeral instance root disk into a semi-persistent storage, in so much as it is preserved even though the guest is no longer running. In general though, the expectation is that the root disks are ephemeral so the ability to take a snapshot cannot be assumed. Therefore this operation is not considered mandatory to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`

  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Suspend instance Status: optional.** Suspending an instance can be thought of as roughly equivalent to suspend-to-disk. The instance no longer consumes any RAM or CPUs, with its live running state having been preserved in a file on disk. It can later be restored, at which point it should continue execution where it left off. As with stopping instance CPUs, it suffers from the fact that the guest OS will typically be left with a clock that is no longer telling correct time. For container based virtualization solutions, this operation is particularly technically challenging to implement and is an area of active research. This operation tends to make more sense when thinking of instances as pets, rather than cattle, since with cattle it would be simpler to just terminate the instance instead of suspending. Therefore this operation is considered optional to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Swap block volumes Status: optional.** The swap volume operation is a mechanism for changing running instance so that its attached volume(s) are backed by different storage in the host. An alternative to this would be to simply terminate the existing instance and spawn a new instance with the new storage. In other words this operation is primarily targeted towards the pet use case rather than cattle. Therefore this is considered optional to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`

  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `missing`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Shutdown instance Status: mandatory.** The ability to terminate a virtual machine is required in order for a cloud user to stop utilizing resources and thus avoid indefinitely ongoing billing. Therefore this operation is mandatory to support in drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `complete`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Resume instance CPUs Status: optional.** See notes for the "Stop instance CPUs" operation

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`

  - **Libvirt Parallels CT:** `complete`

- **Auto configure disk Status: optional.** something something, dark side, something something. Unclear just what this is about.

  drivers:

  - **Libvirt KVM (s390x):** `missing`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `missing`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `missing`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Instance disk I/O limits Status: optional.** The ability to set rate limits on virtual disks allows for greater performance isolation between instances running on the same host storage. It is valid to delegate scheduling of I/O operations to the hypervisor with its default settings, instead of doing fine grained tuning. Therefore this is not considered to be an mandatory configuration to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `missing`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Config drive support Status: choice(guest.setup).** The config drive provides an information channel into the guest operating system, to enable configuration of the administrator password, file injection, registration of SSH keys, etc. Since cloud images typically ship with all login methods locked, a mechanism to set the administrator password of keys is required to get login access. Alternatives include the metadata service and disk injection. At least one of the guest setup mechanisms is required to be supported by drivers, in order to enable login access.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Inject files into disk image Status: optional.** This allows for the end user to provide data for multiple files to be injected into the root filesystem before an instance is booted. This requires that the compute node understand the format of the filesystem and any partitioning scheme it might use on the block device. This is a non-trivial problem considering the vast number of filesystems in existence. The problem of injecting files to a guest OS is better solved by obtaining via the metadata service or config drive. Therefore this operation is considered optional to support.

  drivers:

  - **Libvirt KVM (s390x):** `missing`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Inject guest networking config Status: optional.** This allows for static networking configuration (IP address, netmask, gateway and routes) to be injected directly into the root filesystem before an instance is booted. This requires that the compute node understand how networking is configured in the guest OS which is a non-trivial problem considering the vast number of operating system types. The problem of configuring networking is better solved by DHCP or by obtaining static config via the metadata service or config drive. Therefore this operation is considered optional to support.

  drivers:

  - **Libvirt KVM (s390x):** `missing`

  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `partial` Only for Debian derived guests
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `partial` Only for Debian derived guests
  - **XenServer:** `partial` Only for Debian derived guests
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `partial` requires vmware tools installed
  - **Libvirt Parallels CT:** `missing`

- **Remote desktop over RDP Status: choice(console).** This allows the administrator to interact with the graphical console of the guest OS via RDP. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Some operating systems may prefer to emit messages via the serial console for easier consumption. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **Libvirt KVM (s390x):** `missing`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `missing`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `missing`
  - **XenServer:** `missing`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **View serial console logs Status: choice(console).** This allows the administrator to query the logs of data emitted by the guest OS on its virtualized serial port. For UNIX guests this typically includes all boot up messages and so is useful for diagnosing problems when an instance fails to successfully boot. Not all guest operating systems will be able to emit boot information on a serial console, others may only support graphical consoles. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`

  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Remote interactive serial console Status: choice(console).** This allows the administrator to interact with the serial console of the guest OS. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Not all guest operating systems will be able to emit boot information on a serial console, others may only support graphical consoles. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations. This feature was introduced in the Juno release with blueprint https://blueprints.launchpad.net/nova/+spec/serial-ports

  CLI commands:

  - `nova get-serial-console <server>`

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `unknown`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `unknown`
  - **Hyper-V:** `missing` Will be complete when this review is merged: https://review.openstack.org/#/c/145004/
  - **Libvirt Parallels VM:** `unknown`
  - **Libvirt QEMU (x86):** `unknown`
  - **XenServer:** `missing`
  - **Libvirt Xen:** `unknown`
  - **Ironic:** `unknown`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `unknown`

- **Remote desktop over SPICE Status: choice(console).** This allows the administrator to interact with the graphical console of the guest OS via SPICE. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Some operating systems may prefer to emit messages via the serial console for easier consumption. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **Libvirt KVM (s390x):** `missing`

  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `missing`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `missing`
  - **Libvirt Xen:** `missing`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Remote desktop over VNC Status: choice(console).** This allows the administrator to interact with the graphical console of the guest OS via VNC. This provides a way to see boot up messages and login to the instance when networking configuration has failed, thus preventing a network based login. Some operating systems may prefer to emit messages via the serial console for easier consumption. Therefore support for this operation is not mandatory, however, a driver is required to support at least one of the listed console access operations.

  drivers:

  - **Libvirt KVM (s390x):** `missing`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `missing`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Block storage support Status: optional.** Block storage provides instances with direct attached virtual disks that can be used for persistent storage of data. As an alternative to direct attached disks, an instance may choose to use network based persistent storage. OpenStack provides object storage via the Swift service, or a traditional filesystem such as as NFS/GlusterFS may be used. Some types of instances may not require persistent storage at all, being simple transaction processing systems reading requests & sending results to and from the network. Therefore support for this configuration is not considered mandatory for drivers to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`

  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `partial`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Block storage over fibre channel Status: optional.** To maximise performance of the block storage, it may be desirable to directly access fibre channel LUNs from the underlying storage technology on the compute hosts. Since this is just a performance optimization of the I/O path it is not considered mandatory to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `missing`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `missing`

- **Block storage over iSCSI Status: condition(storage.block==complete).** If the driver wishes to support block storage, it is common to provide an iSCSI based backend to access the storage from cinder. This isolates the compute layer for knowledge of the specific storage technology used by Cinder, albeit at a potential performance cost due to the longer I/O path involved. If the driver chooses to support block storage, then this is considered mandatory to support, otherwise it is considered optional.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`

  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **CHAP authentication for iSCSI Status: optional.** If accessing the cinder iSCSI service over an untrusted LAN it is desirable to be able to enable authentication for the iSCSI protocol. CHAP is the commonly used authentication protocol for iSCSI. This is not considered mandatory to support. (?)

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `missing`

- **Image storage support Status: mandatory.** This refers to the ability to boot an instance from an image stored in the glance image repository. Without this feature it would not be possible to bootstrap from a clean environment, since there would be no way to get block volumes populated and reliance on external PXE servers is out of scope. Therefore this is considered a mandatory storage feature to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `complete`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Network firewall rules Status: optional.** Unclear how this is different from security groups

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `missing`
  - **Libvirt Parallels CT:** `complete`

- **Network routing Status: optional.** Unclear what this refers to

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `missing`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `complete`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **Network security groups Status: optional.** The security groups feature provides a way to define rules to isolate the network traffic of different instances running on a compute host. This would prevent actions such as MAC and IP address spoofing, or the ability to setup rogue DHCP servers. In a private cloud environment this may be considered to be a superfluous requirement. Thereforce this is considered to be an optional configuration to support.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`

  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `missing`
  - **VMware vCenter:** `partial` This is supported by the Neutron NSX plugins
  - **Libvirt Parallels CT:** `complete`

- **Flat networking Status: choice(networking.topology).** Provide network conenctivity to guests using a flat topology across all compute nodes. At least one of the networking configurations is mandatory to support in the drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `complete`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`
  - **Libvirt Xen:** `complete`
  - **Ironic:** `complete`
  - **VMware vCenter:** `complete`
  - **Libvirt Parallels CT:** `complete`

- **VLAN networking Status: choice(networking.topology).** Provide network connectivity to guests using VLANs to define the topology. At least one of the networking configurations is mandatory to support in the drivers.

  drivers:

  - **Libvirt KVM (s390x):** `complete`
  - **Libvirt KVM (ppc64):** `complete`
  - **Libvirt KVM (x86):** `complete`
  - **Libvirt LXC:** `complete`
  - **Hyper-V:** `missing`
  - **Libvirt Parallels VM:** `complete`
  - **Libvirt QEMU (x86):** `complete`
  - **XenServer:** `complete`

- **Libvirt Xen:** `complete`
- **Ironic:** `missing`
- **VMware vCenter:** `complete`
- **Libvirt Parallels CT:** `complete`

CHAPTER

THREE

DEVELOPER GUIDE

3.1 Introduction

3.1.1 Nova System Architecture

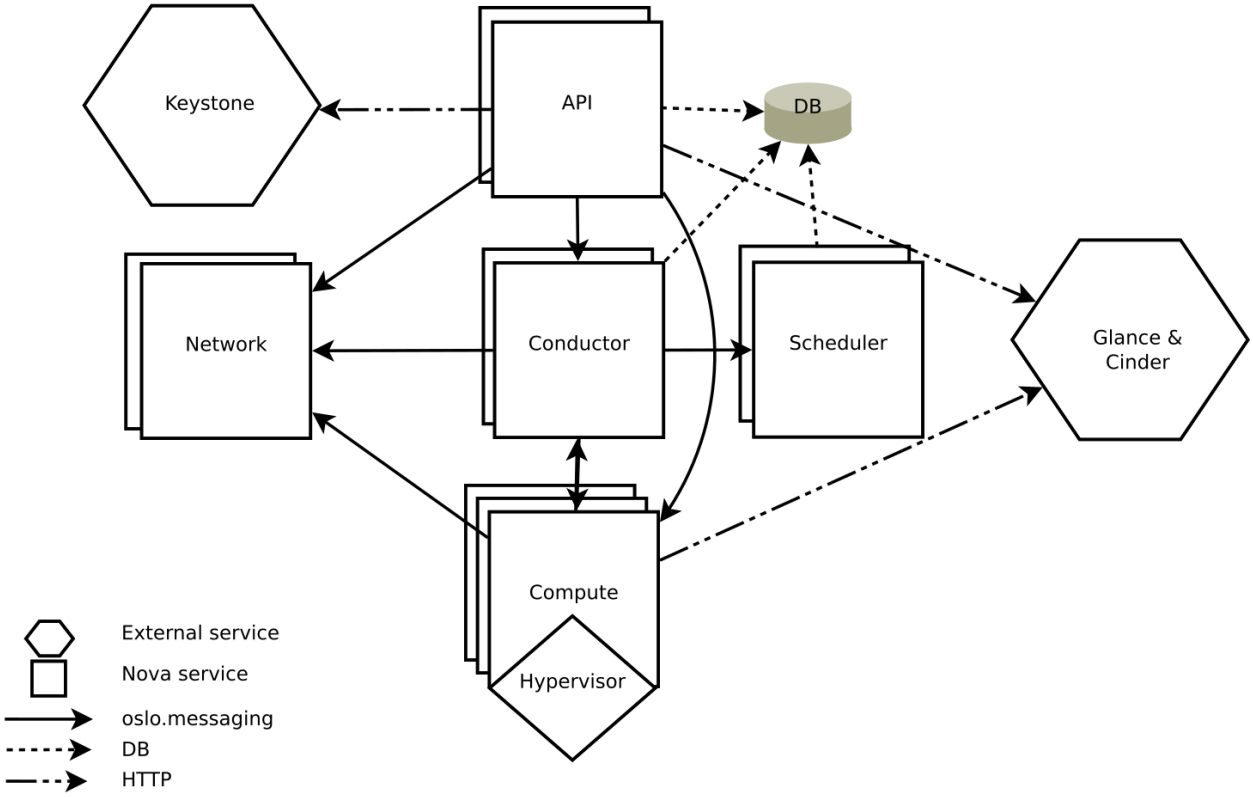
Nova is built on a shared-nothing, messaging-based architecture. All of the major nova components can be run on multiple servers. This means that most component to component communication must go via message queue. In order to avoid blocking each component while waiting for a response, we use deferred objects, with a callback that gets triggered when a response is received.

Nova recently moved to using a sql-based central database that is shared by all components in the system. The amount and depth of the data fits into a sql database quite well. For small deployments this seems like an optimal solution. For larger deployments, and especially if security is a concern, nova will be moving towards multiple data stores with some kind of aggregation system.

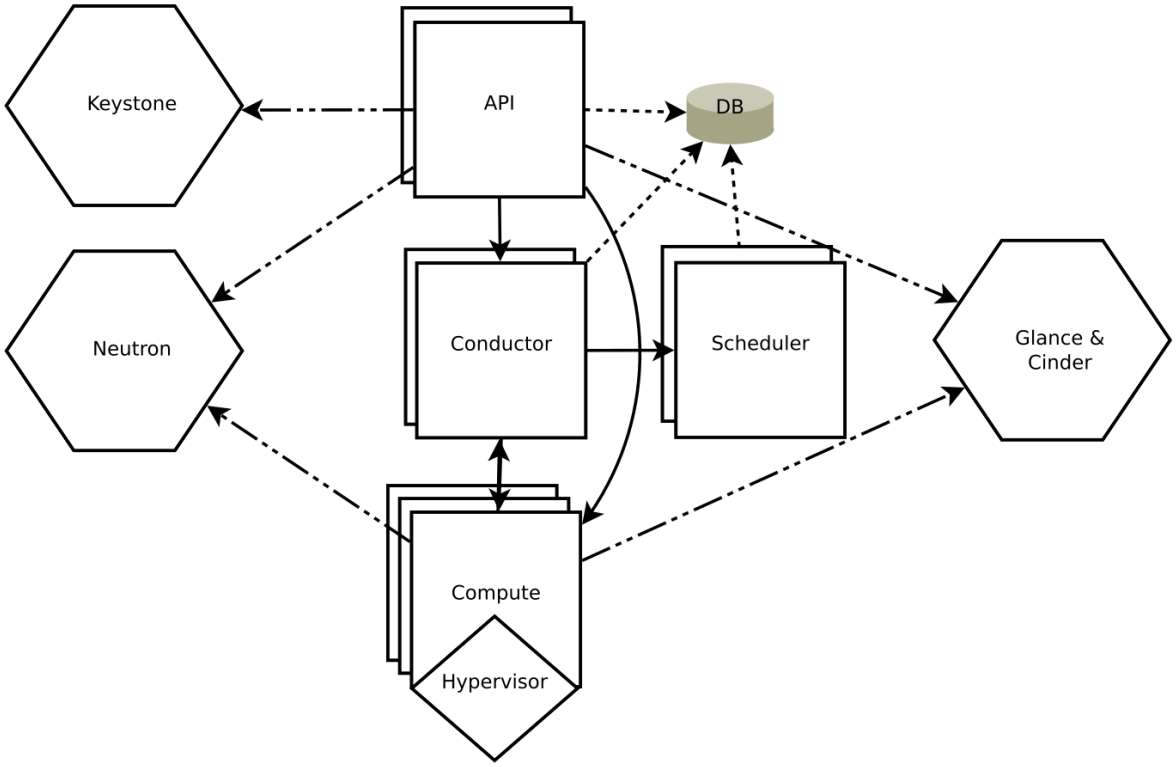
Components

Below you will find a helpful explanation of the different components.

Nova-Networking



Neutron



- DB: sql database for data storage.
- API: component that receives HTTP requests, converts commands and communicates with other components via the **oslo.messaging** queue or HTTP
- Scheduler: decides which host gets each instance
- Network: manages ip forwarding, bridges, and vlans
- Compute: manages communication with hypervisor and virtual machines.
- Conductor: handles requests that need coordination(build/resize), acts as a database proxy, or handles object conversions.

While all services are designed to be horizontally scalable, you should have significantly more computes then anything else.

3.1.2 Scope of the Nova project

Nova is focusing on doing an awesome job of its core mission. This document aims to clarify that core mission.

This is a living document to help record where we agree about what Nova should and should not be doing, and why. Please treat this as a discussion of interesting, and hopefully useful, examples. It is not intended to be an exhaustive policy statement.

Mission

Our mission statement starts with:

> To implement services and associated libraries to provide massively scalable, on demand, self service access to compute resources.

Our official mission statement also includes the following examples of compute resources: bare metal, virtual machines, and containers. For the full official mission statement see: http://governance.openstack.org/reference/projects/nova.html#mission

This document aims to help clarify what the mission statement means.

Compute Resources

Nova is all about access to compute resources. This section looks at the types of compute resource Nova works with.

Virtual Servers

Nova was originally focused purely on providing access to virtual servers running on a variety of different hypervisors. The majority of users use Nova only to provide access to virtual servers from a single hypervisor, however, its possible to have a Nova deployment include multiple different types of hypervisors, while at the same time offering containers and bare metal servers.

Containers

The Nova API is not a good fit for a lot of container use cases. The Magnum project intends to deliver a good container experience built on top of Nova.

Nova allows you to use containers in a similar way to how you would use on demand virtual machines. We want to maintain this distinction, so we maintain the integrity and usefulness of the existing Nova API.

For example, Nova is not designed to spin up new containers for every apache request, nor do we plan to control what goes on inside containers. They get the same metadata provided to them as virtual machines, to do with as they see fit.

Bare Metal Servers

Ironic project has been pioneering the idea of treating physical machines in a similar way to on demand virtual machines.

Nova's driver is able to allow a multi-tenant cloud style use of Ironic controlled resources.

While currently there are operations that are a fundamental part of our virtual machine abstraction that are not currently available in ironic, such as attaching iSCSI volumes, it does not fundamentally change the semantics of our API, and as such is a suitable Nova driver. Moreover, it is expected that gap with shrink over time.

Driver Parity

Our goal for the Nova API to provide a consistent abstraction to access on demand compute resources. We are not aiming to expose all features of all hypervisors. Where the details of the underlying hypervisor leak through our APIs, we have failed in this goal, and we must work towards better abstractions that are more interoperable. This is one reason why we put so much emphasis on the use of Tempest in third party CI systems.

The key tenant of driver parity is that if a feature is supported in a driver, it must feel the same to users, as if they where using any of the other drivers that also support that feature. The exception is that, if possible for widely different performance characteristics, but the effect of that API call must be identical.

Following on from that, should a feature only be added to one of the drivers, we must make every effort to ensure another driver could be implemented to match that behavior.

Its important that drivers support enough features, so the API actually provides a consistent abstraction. For example, being unable to create a server or delete a server, would severely undermine that goal. In fact, Nova only ever manages resources it creates.

Upgrades

Nova is widely used in production. As such we need to respect the needs of our existing users. At the same time we need evolve the current code base, including both adding and removing features.

This section outlines how we expect people to upgrade, and what we do to help existing users that upgrade in the way we expect.

Upgrade expectations

Our upgrade plan is to concentrate on upgrades from N-1 to the Nth release. So for someone running juno, they would have to upgrade to kilo before upgrading to liberty. This is designed to balance the need for a smooth upgrade, against having to keep maintaining the compatibility code to make that upgrade possible. We talk about this approach as users consuming the stable branch.

In addition, we also support users upgrading from the master branch. Technically, between any two between any two commits within the same release cycle. In certain cases, when crossing release boundaries, you must upgrade to the stable branch, before then upgrading to the tip of master. This is to support those that are doing some level of "Continuous Deployment" from the tip of master into production. Many of the public cloud provides running

OpenStack use this approach so they are able to get access to bug fixes and features they work on into production sooner.

This becomes important when you consider reverting a commit that turns out to have been bad idea. We have to assume any public API change may have already been deployed into production, and as such cannot be reverted. In a similar way, a database migration may have been deployed.

Any commit that will affect an upgrade gets the UpgradeImpact tag added to the commit message, so there is no requirement to wait for release notes.

Don't break existing users

As a community we are aiming towards a smooth upgrade process, where users must be unaware you have just upgraded your deployment, except that there might be additional feature available and improved stability and performance of some existing features.

We don't ever want to remove features our users rely on. Sometimes we need to migrate users to a new implementation of that feature, which may require extra steps by the deployer, but the end users must be unaffected by such changes. However there are times when some features become a problem to maintain, and fall into disrepair. We aim to be honest with our users and highlight the issues we have, so we are in a position to find help to fix that situation. Ideally we are able to rework the feature so it can be maintained, but in some rare cases, the feature no longer works, is not tested, and no one is stepping forward to maintain that feature, the best option can be to remove that feature.

When we remove features, we need warn users by first marking those features as deprecated, before we finally remove the feature. The idea is to get feedback on how important the feature is to our user base. Where a feature is important we work with the whole community to find a path forward for those users.

API Scope

Nova aims to provide a highly interoperable and stable REST API for our users to get self-service access to compute resources.

No more API Proxies

Nova API current has some APIs that are now (in kilo) mostly just a proxy to other OpenStack services. If it were possible to remove a public API, these are some we might start with. As such, we don't want to add any more.

The first example is the API that is a proxy to the Glance v1 API. As Glance moves to deprecate its v1 API, we need to translate calls from the old v1 API we expose, to Glance's v2 API.

The next API to mention is the networking APIs, in particular the security groups API. If you are using nova-network, Nova is still the only way to perform these network operations. But if you use Neutron, security groups has a much richer Neutron API, and if you use both Nova API and Neutron API, the miss match can lead to some very unexpected results, in certain cases.

Our intention is to avoid adding to the problems we already have in this area.

No more Orchestration

Nova is a low level infrastructure API. It is plumbing upon which richer ideas can be built. Heat and Magnum being great examples of that.

While we have some APIs that could be considered orchestration, and we must continue to maintain those, we do not intend to add any more APIs that do orchestration.

Third Party APIs

Nova aims to focus on making a great API that is highly interoperable across all Nova deployments.

We have historically done a very poor job of implementing and maintaining compatibility with third party APIs inside the Nova tree.

As such, all new efforts should instead focus on external projects that provide third party compatibility on top of the Nova API. Where needed, we will work this those projects to extending the Nova API such that its possible to add that functionality on top of the Nova API. However, we do not intend to add API calls for those services to persist third party API specific information in the Nova database. Instead we want to focus on additions that enhance the existing Nova API.

Scalability

Our mission includes the text "massively scalable". Lets discuss what that means.

Nova has three main axises of scale: Number of API requests, number of compute nodes and number of active instances. In many cases the number of compute nodes and active instances are so closely related, you rarely need to consider those separately. There are other items, such at the number of tenants, and the number of instances per tenant. But, again, these are very rarely the key scale issue. Its possible to have a small cloud with lots of requests for very short lived VMs, or a large cloud with lots of longer lived VMs. These need to scale out different components of the Nova system to reach their required level of scale.

Ideally all Nova components are either scaled out to match the number of API requests and build requests, or scaled out to match the number of running servers. If we create components that have their load increased relative to both of these items, we can run into inefficiencies or resource contention. Although it is possible to make that work in some cases, this should always be considered.

We intend Nova to be usable for both small and massive deployments. Where small involves 1-10 hypervisors and massive deployments are single regions with greater than 10,000 hypervisors. That should be seen as our current goal not an upper limit.

There are some features that would not scale well for either the small scale or the very large scale. Ideally we would not accept these features, but if there is a strong case to add such features, we must work hard to ensure you can run without that feature at the scale you are required to run.

IaaS not Batch Processing

Currently Nova focuses on providing on-demand compute resources in the style of classic Infrastructure-as-a-service clouds. A large pool of compute resources that people can consume in a self-service way.

Nova is not currently optimized for dealing with a larger number of requests for compute resources compared with the amount of compute resource thats currently available. We generally assume a level of spare capacity is maintained for future requests. This is needed for users that want to quickly scale out, and extra capacity becomes available again as users scale in. While spare capacity is also not required, we are not optimizing for a system that aims to run at 100% capacity at all times. As such our quota system is more focused on limiting the current level of resource usage, rather than ensuring a fair balance of resources between all incoming requests. This doesn't exclude adding features to support making a better use of spare capacity, such as "spot instances".

There have been discussions around how to change Nova to work better for batch job processing. But the current focus is on how to layer such an abstraction on top of the basic primitives Nova currently provides, possibly adding additional APIs where that makes good sense. Should this turn out to be impractical, we may have to revise our approach.

Deployment and Packaging

Nova does not plan on creating its own packaging or deployment systems.

Our CI infrastructure is powered by Devstack. This can also be used by developers to test their work on a full deployment of Nova.

We do not develop any deployment or packaging for production deployments. Being widely adopted by many distributions and commercial products, we instead choose to work with all those parties to ensure they are able to effectively package and deploy Nova.

3.1.3 Development Quickstart

This page describes how to setup and use a working Python development environment that can be used in developing nova on Ubuntu, Fedora or Mac OS X. These instructions assume you're already familiar with git.

Following these instructions will allow you to build the documentation and run the nova unit tests. If you want to be able to run nova (i.e., launch VM instances), you will also need to — either manually or by letting DevStack do it for you — install libvirt and at least one of the supported hypervisors. Running nova is currently only supported on Linux, although you can run the unit tests on Mac OS X.

Note: For how to contribute to Nova, see HowToContribute. Nova uses the Gerrit code review system, GerritWorkflow.

Setup

There are two ways to create a development environment: using DevStack, or explicitly installing and cloning just what you need.

Using DevStack

See Devstack Documentation. If you would like to use Vagrant, there is a Vagrant for DevStack.

Explicit Install/Clone

DevStack installs a complete OpenStack environment. Alternatively, you can explicitly install and clone just what you need for Nova development.

The first step of this process is to install the system (not Python) packages that are required. Following are instructions on how to do this on Linux and on the Mac.

Linux Systems

Note: This section is tested for Nova on Ubuntu (14.04-64) and Fedora-based (RHEL 6.1) distributions. Feel free to add notes and change according to your experiences or operating system.

Install the prerequisite packages.

On Ubuntu:

```
sudo apt-get install python-dev libssl-dev python-pip git-core libxml2-dev libxslt-dev pkg-config lib
```

On Fedora-based distributions (e.g., Fedora/RHEL/CentOS/Scientific Linux):

```
sudo yum install python-devel openssl-devel python-pip git gcc libxslt-devel mysql-devel postgresql-d
sudo pip-python install tox
```

On openSUSE-based distributions (SLES 12, openSUSE 13.1, Factory or Tumbleweed):

```
sudo zypper in gcc git libffi-devel libmysqlclient-devel libvirt-devel libxslt-devel postgresql-devel
```

Mac OS X Systems Install virtualenv:

```
sudo easy_install virtualenv
```

Check the version of OpenSSL you have installed:

```
openssl version
```

The stock version of OpenSSL that ships with Mac OS X 10.6 (OpenSSL 0.9.8l) or Mac OS X 10.7 (OpenSSL 0.9.8r) or Mac OS X 10.10.3 (OpenSSL 0.9.8zc) works fine with nova. OpenSSL versions from brew like OpenSSL 1.0.1k work fine as well.

Getting the code Once you have the prerequisite system packages installed, the next step is to clone the code.

Grab the code from git:

```
git clone https://git.openstack.org/openstack/nova
cd nova
```

Building the Documentation

Install the prerequisite packages: graphviz

To do a full documentation build, issue the following command while the nova directory is current.

```
tox -edocs
```

That will create a Python virtual environment, install the needed Python prerequisites in that environment, and build all the documentation in that environment.

Running unit tests

See Running Python Unit Tests.

Using a remote debugger

Some modern IDE such as pycharm (commercial) or Eclipse (open source) support remote debugging. In order to run nova with remote debugging, start the nova process with the following parameters –remote_debug-host <host IP where the debugger is running> –remote_debug-port <port it is listening on>

Before you start your nova process, start the remote debugger using the instructions for that debugger. For pycharm - http://blog.jetbrains.com/pycharm/2010/12/python-remote-debug-with-pycharm/ For Eclipse - http://pydev.org/manual_adv_remote_debugger.html

More detailed instructions are located here - http://novaremotedebug.blogspot.com

Using fake computes for tests

The number of instances supported by fake computes is not limited by physical constraints. It allows you to perform stress tests on a deployment with few resources (typically a laptop). But you must avoid using scheduler filters limiting the number of instances per compute (like RamFilter, DiskFilter, AggregateCoreFilter), otherwise they will limit the number of instances per compute.

Fake computes can also be used in multi hypervisor-type deployments in order to take advantage of fake and "real" computes during tests:

- create many fake instances for stress tests
- create some "real" instances for functional tests

Fake computes can be used for testing Nova itself but also applications on top of it.

3.2 APIs Development

3.2.1 Adding a Method to the OpenStack API

The interface is a mostly RESTful API. REST stands for Representational State Transfer and provides an architecture "style" for distributed systems using HTTP for transport. Figure out a way to express your request and response in terms of resources that are being created, modified, read, or destroyed.

Routing

To map URLs to controllers+actions, OpenStack uses the Routes package, a clone of Rails routes for Python implementations. See http://routes.groovie.org/ for more information.

URLs are mapped to "action" methods on "controller" classes in `nova/api/openstack/__init__/ApiRouter.__init__` .

See http://routes.groovie.org/manual.html for all syntax, but you'll probably just need these two:

- mapper.connect() lets you map a single URL to a single action on a controller.
- mapper.resource() connects many standard URLs to actions on a controller.

Controllers and actions

Controllers live in `nova/api/openstack`, and inherit from nova.wsgi.Controller.

See `nova/api/openstack/compute/servers.py` for an example.

Action methods take parameters that are sucked out of the URL by mapper.connect() or .resource(). The first two parameters are self and the WebOb request, from which you can get the req.environ, req.body, req.headers, etc.

Serialization

Actions return a dictionary, and wsgi.Controller serializes that to JSON or XML based on the request's content-type.

If you define a new controller, you'll need to define a `_serialization_metadata` attribute on the class, to tell wsgi.Controller how to convert your dictionary to XML. It needs to know the singular form of any list tag (e.g. `<servers>` list contains `<server>` tags) and which dictionary keys are to be XML attributes as opposed to subtags (e.g. `<server id="4"/>` instead of `<server><id>4</id></server>`).

See *nova/api/openstack/compute/servers.py* for an example.

Faults

If you need to return a non-200, you should return faults.Fault(webob.exc.HTTPNotFound()) replacing the exception as appropriate.

3.2.2 API Plugins

Background

Nova has two API plugin frameworks, one for the original V2 API and one for what we call V2.1 which also supports V2.1 microversions. The V2.1 API acts from a REST API user point of view in an identical way to the original V2 API. V2.1 is implemented in the same framework as microversions, with the version requested being 2.1.

The V2 API is now frozen and with the exception of significant bugs no change should be made to the V2 API code. API changes should only be made through V2.1 microversions.

This document covers how to write plugins for the v2.1 framework. A microversions specific document covers the details around what is required for the microversions part. It does not cover V2 plugins which should no longer be developed.

There may still be references to a v3 API both in comments and in the directory path of relevant files. This is because v2.1 first started out being called v3 rather than v2.1. Where you see references to v3 you can treat it as a reference to v2.1 with or without microversions support.

The original V2 API plugins live in `nova/api/openstack/compute/contrib` and the V2.1 plugins live in `nova/api/openstack/compute/plugins/v3`.

Note that any change to the Nova API to be merged will first require a spec be approved first. See here for the appropriate repository. For guidance on the design of the API please refer to the Openstack API WG

Basic plugin structure

A very basic skeleton of a v2.1 plugin can be seen here in the unittests. An annotated version below:

```
"""Basic Test Extension"""

from nova.api.openstack import extensions
from nova.api.openstack import wsgi


ALIAS = 'test-basic'
# ALIAS needs to be unique and should be of the format
# ^[a-z]+[a-z\-]*[a-z]$

class BasicController(wsgi.Controller):

    # Define support for GET on a collection
    def index(self, req):
        data = {'param': 'val'}
        return data

    # Defining a method implements the following API responses:
    #   delete -> DELETE
    #   update -> PUT
```

```
    #   create -> POST
    #   show -> GET
    # If a method is not definied a request to it will be a 404 response

    # It is also possible to define support for further responses
    # See `servers.py <http://git.openstack.org/cgit/openstack/nova/tree/nova/nova/api/openstack/comp

class Basic(extensions.V3APIExtensionBase):
    """Basic Test Extension."""

    name = "BasicTest"
    alias = ALIAS
    version = 1

    # Both get_resources and get_controller_extensions must always
    # be definied by can return an empty array
    def get_resources(self):
        resource = extensions.ResourceExtension('test', BasicController())
        return [resource]

    def get_controller_extensions(self):
        return []
```

All of these plugin files should live in the `nova/api/openstack/compute/plugins/v3` directory.

Policy

Policy (permission) is defined `etc/nova/policy.json`. Implementation of policy is changing a bit at the moment. Will add more to this document or reference another one in the future. Note that a 'discoverable' policy needs to be added for each plugin that you wish to appear in the `/extension` output. Also look at the authorize call in plugins currently merged.

Modularity

The Nova REST API is separated into different plugins in the directory 'nova/api/openstack/compute/plugins/v3/'

Because microversions are supported in the Nova REST API, the API can be extended without any new plugin. But for code readability, the Nova REST API code still needs modularity. Here are rules for how to separate modules:

- You are adding a new resource The new resource should be in standalone module. There isn't any reason to put different resources in a single module.
- Add sub-resource for existing resource To prevent an existing resource module becoming over-inflated, the sub-resource should be implemented in a separate module.
- Add extended attributes for existing resource In normally, the extended attributes is part of existing resource's data model too. So this can be added into existing resource module directly and lightly. To avoid namespace complexity, we should avoid to add extended attributes in existing extended models. New extended attributes needn't any namespace prefix anymore.

Support files

At least one entry needs to made in `setup.cfg` for each plugin. An entry point for the plugin must be added to nova.api.v3.extensions even if no resource or controller is added. Other entry points available are

- Modify create behaviour (nova.api.v3.extensions.server.create)
- Modify rebuild behaviour (nova.api.v3.extensions.server.rebuild)
- Modify update behaviour (nova.api.v3.extensions.server.update)
- Modify resize behaviour (nova.api.v3.extensions.server.resize)

These are essentially hooks into the servers plugin which allow other plugins to modify behaviour without having to modify servers.py. In the past not having this capability led to very large chunks of unrelated code being added to servers.py which was difficult to maintain.

Unit Tests

Should write something more here. But you need to have both unit and functional tests.

Functional tests and API Samples

Should write something here

Commit message tags

Please ensure you add the `DocImpact` tag along with a short description for any API change.

3.2.3 API Microversions

Background

Nova uses a framework we call 'API Microversions' for allowing changes to the API while preserving backward compatibility. The basic idea is that a user has to explicitly ask for their request to be treated with a particular version of the API. So breaking changes can be added to the API without breaking users who don't specifically ask for it. This is done with an HTTP header `X-OpenStack-Nova-API-Version` which is a monotonically increasing semantic version number starting from `2.1`.

If a user makes a request without specifying a version, they will get the `DEFAULT_API_VERSION` as defined in `nova/api/openstack/wsgi.py`. This value is currently `2.1` and is expected to remain so for quite a long time.

There is a special value `latest` which can be specified, which will allow a client to always receive the most recent version of API responses from the server.

For full details please read the Kilo spec for microversions

In Code

In `nova/api/openstack/wsgi.py` we define an `@api_version` decorator which is intended to be used on top-level Controller methods. It is not appropriate for lower-level methods. Some examples:

Adding a new API method

In the controller class:

```
@wsgi.Controller.api_version("2.4")
def my_api_method(self, req, id):
    ....
```

This method would only be available if the caller had specified an `X-OpenStack-Nova-API-Version` of >= `2.4`. If they had specified a lower version (or not specified it and received the default of `2.1`) the server would respond with `HTTP/404`.

Removing an API method

In the controller class:

```
@wsgi.Controller.api_version("2.1", "2.4")
def my_api_method(self, req, id):
    ....
```

This method would only be available if the caller had specified an `X-OpenStack-Nova-API-Version` of <= `2.4`. If `2.5` or later is specified the server will respond with `HTTP/404`.

Changing a method's behaviour

In the controller class:

```
@wsgi.Controller.api_version("2.1", "2.3")
def my_api_method(self, req, id):
    .... method_1 ...

@wsgi.Controller.api_version("2.4")  # noqa
def my_api_method(self, req, id):
    .... method_2 ...
```

If a caller specified `2.1`, `2.2` or `2.3` (or received the default of `2.1`) they would see the result from `method_1`, `2.4` or later `method_2`.

It is vital that the two methods have the same name, so the second of them will need `# noqa` to avoid failing flake8's `F811` rule. The two methods may be different in any kind of semantics (schema validation, return values, response codes, etc)

A method with only small changes between versions

A method may have only small changes between microversions, in which case you can decorate a private method:

```
@api_version("2.1", "2.4")
def _version_specific_func(self, req, arg1):
    pass

@api_version(min_version="2.5")  # noqa
def _version_specific_func(self, req, arg1):
    pass

def show(self, req, id):
    .... common stuff ....
    self._version_specific_func(req, "foo")
    .... common stuff ....
```

A change in schema only

If there is no change to the method, only to the schema that is used for validation, you can add a version range to the `validation.schema` decorator:

```
@wsgi.Controller.api_version("2.1")
@validation.schema(dummy_schema.dummy, "2.3", "2.8")
@validation.schema(dummy_schema.dummy2, "2.9")
def update(self, req, id, body):
    ....
```

This method will be available from version `2.1`, validated according to `dummy_schema.dummy` from `2.3` to `2.8`, and validated according to `dummy_schema.dummy2` from `2.9` onward.

When not using decorators

When you don't want to use the `@api_version` decorator on a method or you want to change behaviour within a method (say it leads to simpler or simply a lot less code) you can directly test for the requested version with a method as long as you have access to the api request object (commonly called `req`). Every API method has an api_version_request object attached to the req object and that can be used to modify behaviour based on its value:

```
def index(self, req):
    <common code>

    req_version = req.api_version_request
    if req_version.matches("2.1", "2.5"):
        ....stuff....
    elif req_version.matches("2.6", "2.10"):
        ....other stuff....
    elif req_version > api_version_request.APIVersionRequest("2.10"):
        ....more stuff.....

    <common code>
```

The first argument to the matches method is the minimum acceptable version and the second is maximum acceptable version. A specified version can be null:

```
null_version = APIVersionRequest()
```

If the minimum version specified is null then there is no restriction on the minimum version, and likewise if the maximum version is null there is no restriction the maximum version. Alternatively a one sided comparison can be used as in the example above.

Other necessary changes

If you are adding a patch which adds a new microversion, it is necessary to add changes to other places which describe your change:

- Update `REST_API_VERSION_HISTORY` in `nova/api/openstack/api_version_request.py`
- Update `_MAX_API_VERSION` in `nova/api/openstack/api_version_request.py`
- Add a verbose description to `nova/api/openstack/rest_api_version_history.rst`. There should be enough information that it could be used by the docs team for release notes.
- Update the expected versions in affected tests, for example in `nova/tests/unit/api/openstack/compute/test_ver`

Allocating a microversion

If you are adding a patch which adds a new microversion, it is necessary to allocate the next microversion number. Except under extremely unusual circumstances and this would have been mentioned in the nova spec for the change, the minor number of `_MAX_API_VERSION` will be incremented. This will also be the new microversion number for the API change.

It is possible that multiple microversion patches would be proposed in parallel and the microversions would conflict between patches. This will cause a merge conflict. We don't reserve a microversion for each patch in advance as we don't know the final merge order. Developers may need over time to rebase their patch calculating a new version number as above based on the updated value of `_MAX_API_VERSION`.

Testing Microversioned API Methods

Testing a microversioned API method is very similar to a normal controller method test, you just need to add the `X-OpenStack-Nova-API-Version` header, for example:

```
req = fakes.HTTPRequest.blank('/testable/url/endpoint')
req.headers = {'X-OpenStack-Nova-API-Version': '2.2'}
req.api_version_request = api_version.APIVersionRequest('2.6')

controller = controller.TestableController()

res = controller.index(req)
... assertions about the response ...
```

For many examples of testing, the canonical examples are in `nova/tests/unit/api/openstack/compute/test_microver`

3.2.4 Rest API Policy Enforcement

Here is a vision of how we want policy to be enforced in nova.

Problems with current system

There are several problems for current API policy.

- The permission checking is spread through the various levels of the nova code, also there are some hard-coded permission checks that make some policies not enforceable.
- API policy rules need better granularity. Some of extensions just use one rule for all the APIs. Deployer can't get better granularity control for the APIs.
- More easy way to override default policy settings for deployer. And Currently all the API(EC2, V2, V2.1) rules mix in one policy.conf file.

These are the kinds of things we need to make easier:

1. Operator wants to enable a specific role to access the service API which is not possible because there is currently a hard coded admin check.

2. One policy rule per API action. Having a check in the REST API and a redundant check in the compute API can confuse developers and deployers.

3. Operator can specify different rules for APIs that in same extension.

4. Operator can override the default policy rule easily without mixing his own config and default config in one policy.conf file.

Future of policy enforcement

The generic rule for all the improvement is keep V2 API back-compatible. Because V2 API may be deprecated after V2.1 parity with V2. This can reduce the risk we take. The improvement just for EC2 and V2.1 API. There isn't any user for V2.1, as it isn't ready yet. We have to do change for EC2 API. EC2 API won't be removed like v2 API. If we keep back-compatible for EC2 API also, the old compute api layer checks won't be removed forever. EC2 API is really small than Nova API. It's about 29 APIs without volume and image related(those policy check done by cinder and glance). So it will affect user less.

Enforcement policy at REST API layer

The policy should be only enforced at REST API layer. This is clear for user to know where the policy will be enforced. If the policy spread into multiple layer of nova code, user won't know when and where the policy will be enforced if they didn't have knowledge about nova code.

Remove all the permission checking under REST API layer. Policy will only be enforced at REST API layer.

This will affect the EC2 API and V2.1 API, there are some API just have policy enforcement at Compute/Network API layer, those policy will be move to API layer and renamed.

Removes hard-code permission checks

Hard-coded permission checks make it impossible to supply a configurable policy. They should be removed in order to make nova auth completely configurable.

This will affect EC2 API and Nova V2.1 API. User need update their policy rule to match the old hard-code permission.

For Nova V2 API, the hard-code permission checks will be moved to REST API layer to guarantee it won't break the back-compatibility. That may ugly some hard-code permission check in API layer, but V2 API will be removed once V2.1 API ready, so our choice will reduce the risk.

Port policy.d from oslo-incubator into nova

This feature make deployer can override default policy rule easily. And When nova default policy config changed, deployer only need replace default policy config files with new one. It won't affect his own policy config in other files.

Use different prefix in policy rule name for EC2/V2/V2.1 API

Currently all the APIs(Nova v2/v2.1 API, EC2 API) use same set of policy rules. Especially there isn't obvious mapping between those policy rules and EC2 API. User can know clearly which policy should be configured for specific API.

Nova should provide different prefix for policy rule name that used to group them, and put them in different policy configure file in policy.d

- EC2 API: Use prefix "ec2_api". The rule looks like "ec2_api:[action]"
- Nova V2 API: After we move to V2.1, we needn't spend time to change V2 api rule, and needn't to bother deployer upgrade their policy config. So just keep V2 API poicy rule named as before.
- Nova V2.1 API: We name the policy rule as "os_compute_api:[extension]:[action]". The core API may be changed in the future, so we needn't name them as "compute" or "compute_extension" to distinguish the core or extension API.

This will affect EC2 API and V2.1 API. For EC2 API, it need deployer update their policy config. For V2.1 API, there isn't any user yet, so there won't any effect.

Group the policy rules into different policy files

After group the policy rules for different API, we can separate them into different files. Then deployer will more clear for which rule he can set for specific API. The rules can be grouped as below:

- policy.conf: It only contains the generic rule, like:

::

    "context_is_admin": "role:admin", "admin_or_owner": "is_admin:True or project_id:%(project_id)s", "default": "rule:admin_or_owner",

- policy.d/00-ec2-api.conf: It contains all the policy rules for EC2 API.
- policy.d/00-v2-api.conf: It contains all the policy rules for nova V2 API.
- policy.d/00-v2.1-api.conf: It contains all the policy rules for nova v2.1 API.

The prefix '00-' is used to order the configure file. All the files in policy.d will be loaded by alphabetical order. '00-' means those files will be loaded very early.

Add separated rule for each API in extension

This is for provider better granularity for policy rules. Not just provide policy rule for extension as unit.

This need user to move the policy rule into separated rule for each API.

Enable action level rule override extension level rule

After separated rule for each API in extension, that will increase the work for deployer. So enable extension level rule as default for each API in that extension will ease that a lot. Deployer also can specify one rule for each API to override the extension level rule.

Existed Nova API being restricted

Nova provide default policy rules for all the APIs. Operator should only make the policy rule more permissive. If the Operator make the API to be restricted that make break the existed API user or application. That's kind of back-incompatible. SO Operator can free to add additional permission to the existed API.

3.2.5 Nova Stable REST API

This document describes both the current state of the Nova REST API – as of the Kilo release – and also attempts to describe how the Nova team intends to evolve the REST API's implementation over time and remove some of the cruft that has crept in over the years.

Background

Nova currently includes two distinct frameworks for exposing REST API functionality. Older code is called the "V2 API" and exists in the /nova/api/openstack/compute/contrib/ directory. Newer code is called the "v2.1 API" and exists in the /nova/api/openstack/compute/plugins directory.

The V2 API is the old Nova REST API. It will be replaced by V2.1 API totally. The code tree of V2 API will be removed in the future also.

The V2.1 API is the new Nova REST API with a set of improvements which includes Microversion and standardized validation of inputs using JSON-Schema. Also the V2.1 API is totally backwards compatible with the V2 API (That is the reason we call it as V2.1 API).

Stable API

In the V2 API, there is a concept called 'extension'. An operator can use it to enable/disable part of Nova REST API based on requirements. An end user may query the '/extensions' API to discover what *API functionality* is supported by the Nova deployment.

Unfortunately, because V2 API extensions could be enabled or disabled from one deployment to another – as well as custom API extensions added to one deployment and not another – it was impossible for an end user to know what the OpenStack Compute API actually included. No two OpenStack deployments were consistent, which made cloud interoperability impossible.

API extensions, while not (yet) removed from the V2.1 API, are no longer needed to evolve the REST API, and no new API functionality should use the API extension classes to implement new functionality. Instead, new API functionality should use the microversioning decorators to add or change the REST API.

The extension is considered as two things in the Nova V2.1 API:

- The '/extensions' API

  In the V2 API the user can query it to determine what APIs are supported by the current Nova deployment.

  In V2.1 API, microversions enable us to add new features in backwards- compatible ways. And microversions not only enable us to add new futures by backwards-compatible method, also can be added by appropriate backwards- incompatible method.

  The '/extensions' API is frozen in Nova V2.1 API and will be deprecated in the future.

- The plugin framework

  One of the improvements in the V2.1 API was using stevedore to load Nova REST API extensions instead of old V2 handcrafted extension load mechanism.

  There was an argument that the plugin framework supported extensibility in the Nova API to allow deployers to publish custom API resources.

  We will keep the existing plugin mechanisms in place within Nova but only to enable modularity in the codebase, not to allow extending of the Nova REST API.

As the extension will be removed from Nove V2.1 REST API. So the concept of core API and extension API is eliminated also. There is no difference between Nova V2.1 REST API, all of them are part of Nova stable REST API.

3.3 Concepts

3.3.1 Host Aggregates

Host aggregates can be regarded as a mechanism to further partition an availability zone; while availability zones are visible to users, host aggregates are only visible to administrators. Host aggregates started out as a way to use Xen hypervisor resource pools, but has been generalized to provide a mechanism to allow administrators to assign key-value pairs to groups of machines. Each node can have multiple aggregates, each aggregate can have multiple key-value pairs, and the same key-value pair can be assigned to multiple aggregate. This information can be used in the scheduler to enable advanced scheduling, to set up xen hypervisor resources pools or to define logical groups for migration.

Xen Pool Host Aggregates

Originally all aggregates were Xen resource pools, now an aggregate can be set up as a resource pool by giving the aggregate the correct key-value pair.

You can use aggregates for XenServer resource pools when you have multiple compute nodes installed (only XenServer/XCP via xenapi driver is currently supported), and you want to leverage the capabilities of the underlying hypervisor resource pools. For example, you want to enable VM live migration (i.e. VM migration within the pool) or enable host maintenance with zero-downtime for guest instances. Please, note that VM migration across pools (i.e. storage migration) is not yet supported in XenServer/XCP, but will be added when available. Bear in mind that the two migration techniques are not mutually exclusive and can be used in combination for a higher level of flexibility in your cloud management.

Design

The OSAPI Admin API is extended to support the following operations:

- Aggregates
  - list aggregates: returns a list of all the host-aggregates (optionally filtered by availability zone)
  - create aggregate: creates an aggregate, takes a friendly name, etc. returns an id
  - show aggregate: shows the details of an aggregate (id, name, availability_zone, hosts and metadata)
  - update aggregate: updates the name and availability zone of an aggregate
  - set metadata: sets the metadata on an aggregate to the values supplied
  - delete aggregate: deletes an aggregate, it fails if the aggregate is not empty
  - add host: adds a host to the aggregate
  - remove host: removes a host from the aggregate
- Hosts
  - start host maintenance (or evacuate-host): disallow a host to serve API requests and migrate instances to other hosts of the aggregate
  - stop host maintenance: (or rebalance-host): put the host back into operational mode, migrating instances back onto that host

Using the Nova CLI

Using the nova command you can create, delete and manage aggregates. The following section outlines the list of available commands.

Usage

```
* aggregate-list                                                Print a list of all aggregates.
* aggregate-create          <name> <availability_zone>          Create a new aggregate with the s
* aggregate-delete          <id>                                Delete the aggregate by its id.
* aggregate-details         <id>                                Show details of the specified agg
* aggregate-add-host        <id> <host>                         Add the host to the specified agg
* aggregate-remove-host     <id> <host>                         Remove the specified host from th
* aggregate-set-metadata    <id> <key=value> [<key=value> ...]  Update the metadata associated wi
* aggregate-update          <id> <name> [<availability_zone>]   Update the aggregate's name and o

* host-list                                                     List all hosts by service
* host-update               --maintenance [enable | disable]    Put/resume host into/from mainter
```

3.3.2 Cells V2

Manifesto

Problem

Nova currently depends on a single logical database and message queue that all nodes depend on for communication and data persistence. This becomes an issue for deployers as scaling and providing fault tolerance for these systems is difficult.

We have an experimental feature in Nova called "cells" which is used by some large deployments to partition compute nodes into smaller groups, coupled with a database and queue. This seems to be a well-liked and easy-to-understand arrangement of resources, but the implementation of it has issues for maintenance and correctness.

Proposal

Right now, when a request hits the Nova API for a particular instance, the instance information is fetched from the database, which contains the hostname of the compute node on which the instance currently lives. If the request needs to take action on the instance (which is most of them), the hostname is used to calculate the name of a queue, and a message is written there which finds its way to the proper compute node.

The meat of this proposal is changing the above hostname lookup into two parts that yield three pieces of information instead of one. Basically, instead of merely looking up the *name* of the compute node on which an instance lives, we will also obtain database and queue connection information. Thus, when asked to take action on instance $foo, we will:

1. Lookup the three-tuple of (database, queue, hostname) for that instance
2. Connect to that database and fetch the instance record
3. Connect to the queue and send the message to the proper hostname queue

The above differs from the current organization in two ways. First, we need to do two database lookups before we know where the instance lives. Second, we need to demand-connect to the appropriate database and queue. Both of these have performance implications, but we believe we can mitigate the impacts through the use of things like a

memcache of instance mapping information and pooling of connections to database and queue systems. The number of cells will always be much smaller than the number of instances.

There are availability implications with this change since something like a 'nova list' which might query multiple cells could end up with a partial result if there is a database failure in a cell. A database failure within a cell would cause larger issues than a partial list result so the expectation is that it would be addressed quickly and cellsv2 will handle it by indicating in the response that the data may not be complete.

Since this is very similar to what we have with current cells, in terms of organization of resources, we have decided to call this "cellsv2" for disambiguation.

After this work is complete there will no longer be a "no cells" deployment. The default installation of Nova will be a single cell setup.

Benefits

The benefits of this new organization are:

- Native sharding of the database and queue as a first-class-feature in nova. All of the code paths will go through the lookup procedure and thus we won't have the same feature parity issues as we do with current cells.
- No high-level replication of all the cell databases at the top. The API will need a database of its own for things like the instance index, but it will not need to replicate all the data at the top level.
- It draws a clear line between global and local data elements. Things like flavors and keypairs are clearly global concepts that need only live at the top level. Providing this separation allows compute nodes to become even more stateless and insulated from things like deleted/changed global data.
- Existing non-cells users will suddenly gain the ability to spawn a new "cell" from their existing deployment without changing their architecture. Simply adding information about the new database and queue systems to the new index will allow them to consume those resources.
- Existing cells users will need to fill out the cells mapping index, shutdown their existing cells synchronization service, and ultimately clean up their top level database. However, since the high-level organization is not substantially different, they will not have to re-architect their systems to move to cellsv2.
- Adding new sets of hosts as a new "cell" allows them to be plugged into a deployment and tested before allowing builds to be scheduled to them.

Comparison with current cells

In reality, the proposed organization is nearly the same as what we currently have in cells today. A cell mostly consists of a database, queue, and set of compute nodes. The primary difference is that current cells require a nova-cells service that synchronizes information up and down from the top level to the child cell. Additionally, there are alternate code paths in compute/api.py which handle routing messages to cells instead of directly down to a compute host. Both of these differences are relevant to why we have a hard time achieving feature and test parity with regular nova (because many things take an alternate path with cells) and why it's hard to understand what is going on (all the extra synchronization of data). The new proposed cellsv2 organization avoids both of these problems by letting things live where they should, teaching nova to natively find the right db, queue, and compute node to handle a given request.

Database split

As mentioned above there is a split between global data and data that is local to a cell.

The following is a breakdown of what data can uncontroversially considered global versus local to a cell. Missing data will be filled in as consensus is reached on the data that is more difficult to cleanly place. The missing data is mostly concerned with scheduling and networking.

Global (API-level) Tables

instance_types instance_type_projects instance_type_extra_specs quotas project_user_quotas quota_classes quota_usages security_groups security_group_rules security_group_default_rules provider_fw_rules key_pairs migrations networks tags

Cell-level Tables

instances instance_info_caches instance_extra instance_metadata instance_system_metadata instance_faults instance_actions instance_actions_events instance_id_mappings pci_devices block_device_mapping virtual_interfaces

3.3.3 Threading model

All OpenStack services use *green thread* model of threading, implemented through using the Python eventlet and greenlet libraries.

Green threads use a cooperative model of threading: thread context switches can only occur when specific eventlet or greenlet library calls are made (e.g., sleep, certain I/O calls). From the operating system's point of view, each OpenStack service runs in a single thread.

The use of green threads reduces the likelihood of race conditions, but does not completely eliminate them. In some cases, you may need to use the `@lockutils.synchronized(...)` decorator to avoid races.

In addition, since there is only one operating system thread, a call that blocks that main thread will block the entire process.

Yielding the thread in long-running tasks

If a code path takes a long time to execute and does not contain any methods that trigger an eventlet context switch, the long-running thread will block any pending threads.

This scenario can be avoided by adding calls to the eventlet sleep method in the long-running code path. The sleep call will trigger a context switch if there are pending threads, and using an argument of 0 will avoid introducing delays in the case that there is only a single green thread:

```
from eventlet import greenthread
...
greenthread.sleep(0)
```

MySQL access and eventlet

Queries to the MySQL database will block the main thread of a service. This is because OpenStack services use an external C library for accessing the MySQL database. Since eventlet cannot use monkey-patching to intercept blocking calls in a C library, the resulting database query blocks the thread.

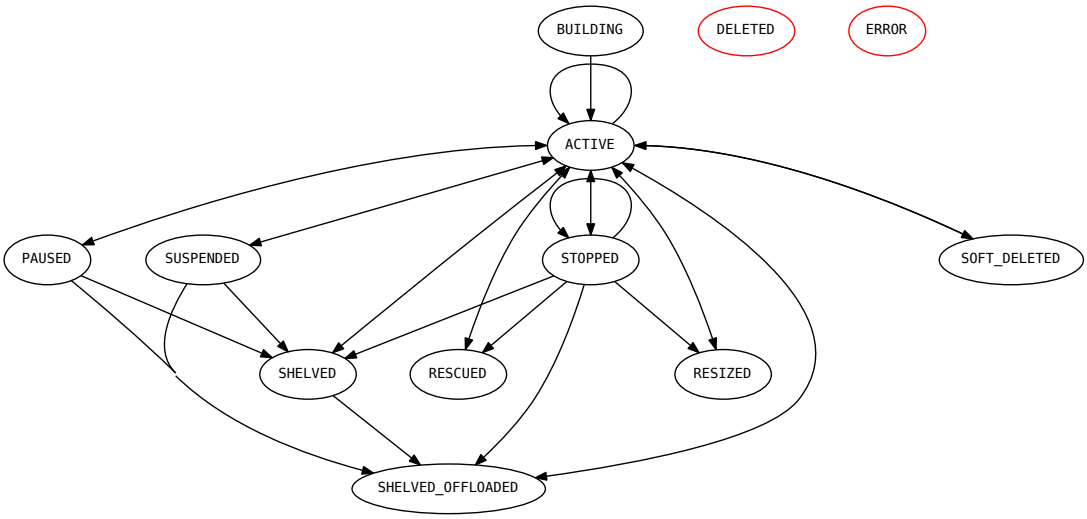
The Diablo release contained a thread-pooling implementation that did not block, but this implementation resulted in a bug and was removed.

See this mailing list thread for a discussion of this issue, including a discussion of the impact on performance.

3.3.4 Virtual Machine States and Transitions

The following diagrams and tables show the required virtual machine (VM) states and task states for various commands issued by the user.

Allowed State Transitions



All states are allowed to transition to DELETED and ERROR.

Requirements for Commands

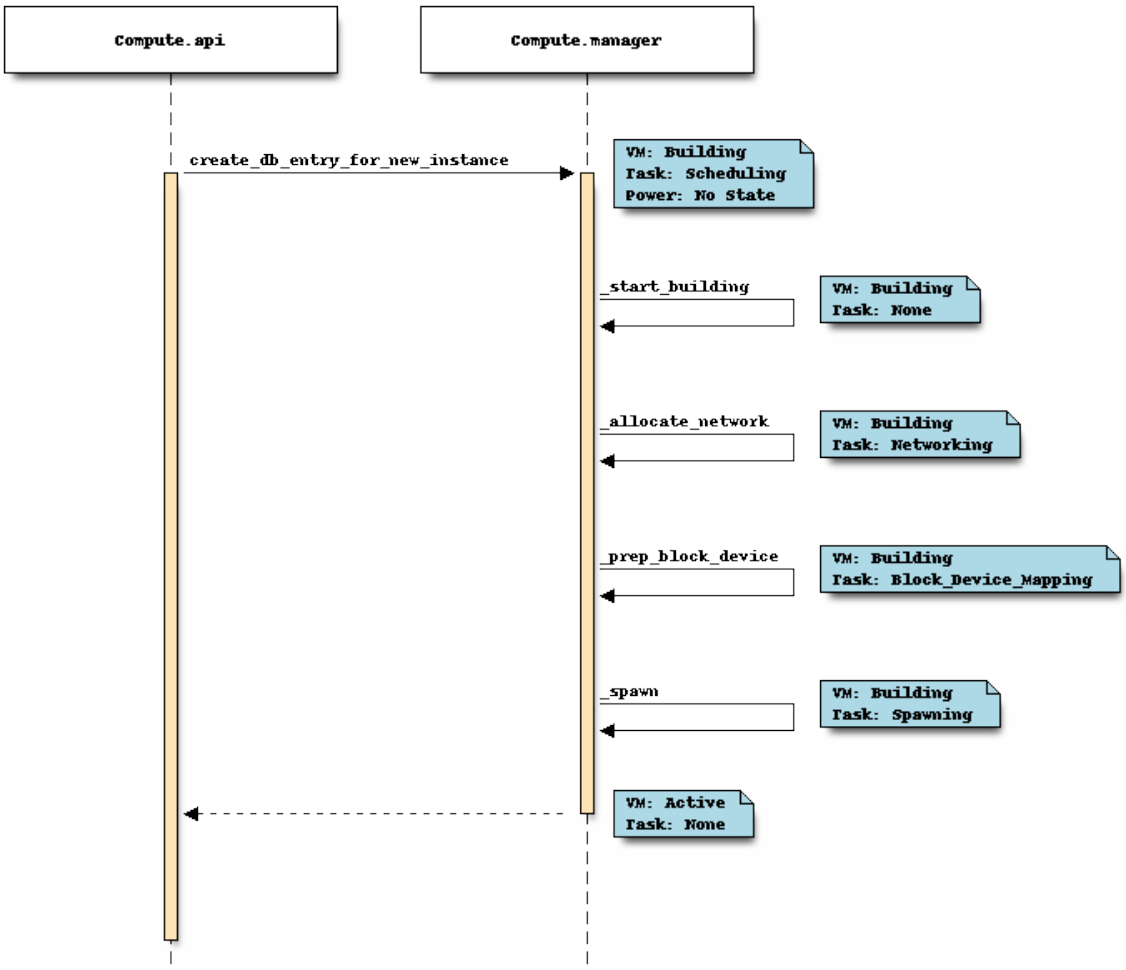
Command	Req'd VM States	Req'd Task States	Target State
pause	Active, Shutoff, Rescued	Resize Verify, unset	Paused
unpause	Paused	N/A	Active
suspend	Active, Shutoff	N/A	Suspended
resume	Suspended	N/A	Active
rescue	Active, Shutoff	Resize Verify, unset	Rescued
unrescue	Rescued	N/A	Active
set admin password	Active	N/A	Active
rebuild	Active, Shutoff	Resize Verify, unset	Active
force delete	Soft Deleted	N/A	Deleted
restore	Soft Deleted	N/A	Active
soft delete	Active, Shutoff, Error	N/A	Soft Deleted
delete	Active, Shutoff, Building, Rescued, Error	N/A	Deleted
backup	Active, Shutoff	N/A	Active, Shutoff
snapshot	Active, Shutoff	N/A	Active, Shutoff
start	Shutoff, Stopped	N/A	Active
stop	Active, Shutoff, Rescued	Resize Verify, unset	Stopped
reboot	Active, Shutoff, Rescued	Resize Verify, unset	Active
resize	Active, Shutoff	Resize Verify, unset	Resized
revert resize	Active, Shutoff	Resize Verify, unset	Active
confirm resize	Active, Shutoff	Resize Verify, unset	Active

VM states and Possible Commands

VM State	Commands
Paused	unpause
Suspended	resume
Active	set admin password, suspend, pause, rescue, rebuild, soft delete, delete, backup, snapshot, stop, reboot, resize, revert resize, confirm resize
Shutoff	suspend, pause, rescue, rebuild, soft delete, delete, backup, start, snapshot, stop, reboot, resize, revert resize, confirm resize
Rescued	unrescue, pause
Stopped	rescue, delete, start
Soft Deleted	force delete, restore
Error	soft delete, delete
Building	delete
Rescued	delete, stop, reboot

Create Instance States

The following diagram shows the sequence of VM states, task states, and power states when a new VM instance is created.



3.3.5 Internationalization

Nova uses the oslo.i18n library to support internationalization. The oslo.i18n library is built on top of gettext and provides functions that are used to enable user-facing strings such as log messages to appear in the appropriate language in different locales.

Nova exposes the oslo.i18n library support via the `nova/i18n.py` integration module. This module provides the functions needed to wrap translatable strings. It provides the `_()` wrapper for general user-facing messages and specific wrappers for messages used only for logging. DEBUG level messages do not need translation but CRITICAL, ERROR, WARNING and INFO messages should be wrapped with `_LC()`, `_LE()`, `_LW()` or `_LI()` respectively.

For example:

```
LOG.debug("block_device_mapping %(mapping)s",
          {'mapping': block_device_mapping})
```

or:

```
LOG.warn(_LW('Unknown base file %(img)s'), {'img': img})
```

You should use the basic wrapper `_()` for strings which are not log messages:

```
raise nova.SomeException(_('Invalid service catalogue'))
```

Do not use `locals()` for formatting messages because: 1. It is not as clear as using explicit dicts. 2. It could produce hidden errors during refactoring. 3. Changing the name of a variable causes a change in the message. 4. It creates a lot of otherwise unused variables.

If you do not follow the project conventions, your code may cause hacking checks to fail.

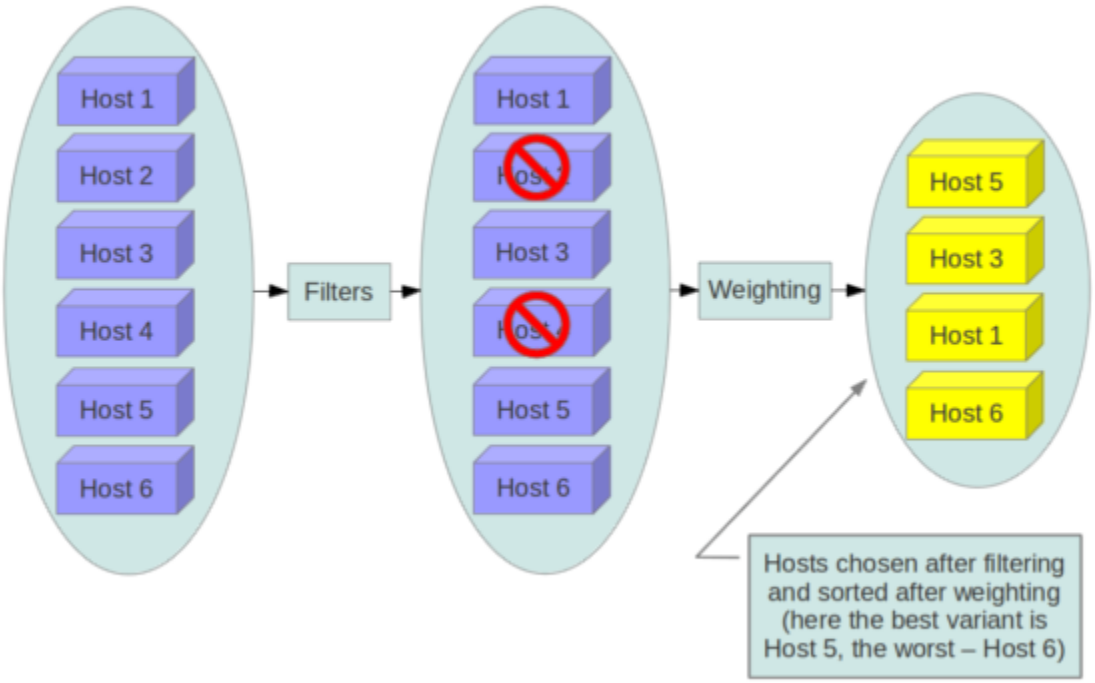
The `_()`, `_LC()`, `_LE()`, `_LW()` and `_LI()` functions can be imported with:

```
from nova.i18n import _
from nova.i18n import _LC
from nova.i18n import _LE
from nova.i18n import _LW
from nova.i18n import _LI
```

3.3.6 Filter Scheduler

The **Filter Scheduler** supports *filtering* and *weighting* to make informed decisions on where a new instance should be created. This Scheduler supports only working with Compute Nodes.

Filtering



During its work Filter Scheduler firstly makes dictionary of unfiltered hosts, then filters them using filter properties and finally chooses hosts for the requested number of instances (each time it chooses the most weighed host and appends it to the list of selected hosts).

If it turns up, that it can't find candidates for the next instance, it means that there are no more appropriate hosts where the instance could be scheduled.

If we speak about *filtering* and *weighting*, their work is quite flexible in the Filter Scheduler. There are a lot of filtering strategies for the Scheduler to support. Also you can even implement *your own algorithm of filtering*.

There are some standard filter classes to use (`nova.scheduler.filters`):

- `AllHostsFilter` - frankly speaking, this filter does no operation. It passes all the available hosts.
- `ImagePropertiesFilter` - filters hosts based on properties defined on the instance's image. It passes hosts that can support the specified image properties contained in the instance.
- `AvailabilityZoneFilter` - filters hosts by availability zone. It passes hosts matching the availability zone specified in the instance properties. Use a comma to specify multiple zones. The filter will then ensure it matches any zone specified.
- `ComputeCapabilitiesFilter` - checks that the capabilities provided by the host compute service satisfy any extra specifications associated with the instance type. It passes hosts that can create the specified instance type.

  If an extra specs key contains a colon (:), anything before the colon is treated as a namespace and anything after the colon is treated as the key to be matched. If a namespace is present and is not `capabilities`, the filter ignores the namespace. Example like `capabilities:cpu_info:features` is a valid scope format. For backward compatibility, also treats the extra specs key as the key to be matched if no namespace is present; this action is highly discouraged because it conflicts with AggregateInstanceExtraSpecsFilter filter when you enable both filters

  The extra specifications can have an operator at the beginning of the value string of a key/value pair. If there is no operator specified, then a default operator of `s==` is used. Valid operators are:

```
* = (equal to or greater than as a number; same as vcpus case)
* == (equal to as a number)
* != (not equal to as a number)
* >= (greater than or equal to as a number)
* <= (less than or equal to as a number)
* s== (equal to as a string)
* s!= (not equal to as a string)
* s>= (greater than or equal to as a string)
* s> (greater than as a string)
* s<= (less than or equal to as a string)
* s< (less than as a string)
* <in> (substring)
* <all-in> (all elements contained in collection)
* <or> (find one of these)

Examples are: ">= 5", "s== 2.1.0", "<in> gcc", "<all-in> aes mmx", and "<or> fpu <or> gpu"
```

- `AggregateInstanceExtraSpecsFilter` - checks that the aggregate metadata satisfies any extra specifications associated with the instance type (that have no scope or are scoped with `aggregate_instance_extra_specs`). It passes hosts that can create the specified instance type. The extra specifications can have the same operators as `ComputeCapabilitiesFilter`. To specify multiple values for the same key use a comma. E.g., "value1,value2"
- `ComputeFilter` - passes all hosts that are operational and enabled.
- `CoreFilter` - filters based on CPU core utilization. It passes hosts with sufficient number of CPU cores.
- `AggregateCoreFilter` - filters hosts by CPU core number with per-aggregate `cpu_allocation_ratio` setting. If no per-aggregate value is found, it will fall back to the global

default `cpu_allocation_ratio`. If more than one value is found for a host (meaning the host is in two different aggregate with different ratio settings), the minimum value will be used.

- `IsolatedHostsFilter` - filter based on `image_isolated`, `host_isolated` and `restrict_isolated_hosts_to_isolated_images` flags.
- `JsonFilter` - allows simple JSON-based grammar for selecting hosts.
- `RamFilter` - filters hosts by their RAM. Only hosts with sufficient RAM to host the instance are passed.
- `AggregateRamFilter` - filters hosts by RAM with per-aggregate `ram_allocation_ratio` setting. If no per-aggregate value is found, it will fall back to the global default `ram_allocation_ratio`. If more than one value is found for a host (meaning the host is in two different aggregate with different ratio settings), the minimum value will be used.
- `DiskFilter` - filters hosts by their disk allocation. Only hosts with sufficient disk space to host the instance are passed. `disk_allocation_ratio` setting. It's virtual disk to physical disk allocation ratio and it's 1.0 by default. The total allow allocated disk size will be physical disk multiplied this ratio.
- `AggregateDiskFilter` - filters hosts by disk allocation with per-aggregate `disk_allocation_ratio` setting. If no per-aggregate value is found, it will fall back to the global default `disk_allocation_ratio`. If more than one value is found for a host (meaning the host is in two or more different aggregates with different ratio settings), the minimum value will be used.
- `NumInstancesFilter` - filters hosts by number of running instances on it. hosts with too many instances will be filtered. `max_instances_per_host` setting. Maximum number of instances allowed to run on this host, the host will be ignored by scheduler if more than `max_instances_per_host` are already existing on the host.
- `AggregateNumInstancesFilter` - filters hosts by number of instances with per-aggregate `max_instances_per_host` setting. If no per-aggregate value is found, it will fall back to the global default `max_instances_per_host`. If more than one value is found for a host (meaning the host is in two or more different aggregates with different max instances per host settings), the minimum value will be used.
- `IoOpsFilter` - filters hosts by concurrent I/O operations on it. hosts with too many concurrent I/O operations will be filtered. `max_io_ops_per_host` setting. Maximum number of I/O intensive instances allowed to run on this host, the host will be ignored by scheduler if more than `max_io_ops_per_host` instances such as build/resize/snapshot etc are running on it.
- `AggregateIoOpsFilter` - filters hosts by I/O operations with per-aggregate `max_io_ops_per_host` setting. If no per-aggregate value is found, it will fall back to the global default `max_io_ops_per_host`. If more than one value is found for a host (meaning the host is in two or more different aggregates with different max io operations settings), the minimum value will be used.
- `PciPassthroughFilter` - Filter that schedules instances on a host if the host has devices to meet the device requests in the 'extra_specs' for the flavor.
- `SimpleCIDRAffinityFilter` - allows to put a new instance on a host within the same IP block.
- `DifferentHostFilter` - allows to put the instance on a different host from a set of instances.
- `SameHostFilter` - puts the instance on the same host as another instance in a set of instances.
- `RetryFilter` - filters hosts that have been attempted for scheduling. Only passes hosts that have not been previously attempted.
- `TrustedFilter` - filters hosts based on their trust. Only passes hosts that meet the trust requirements specified in the instance properties.
- `TypeAffinityFilter` - Only passes hosts that are not already running an instance of the requested type.

- **AggregateTypeAffinityFilter - limits instance_type by aggregate.** This filter passes hosts if no instance_type key is set or the instance_type aggregate metadata value contains the name of the instance_type requested. The value of the instance_type metadata entry is a string that may contain either a single instance_type name or a comma separated list of instance_type names. e.g. 'm1.nano' or "m1.nano,m1.small"
- ServerGroupAntiAffinityFilter - This filter implements anti-affinity for a server group. First you must create a server group with a policy of 'anti-affinity' via the server groups API. Then, when you boot a new server, provide a scheduler hint of 'group=<uuid>' where <uuid> is the UUID of the server group you created. This will result in the server getting added to the group. When the server gets scheduled, anti-affinity will be enforced among all servers in that group.
- ServerGroupAffinityFilter - This filter works the same way as ServerGroupAntiAffinityFilter. The difference is that when you create the server group, you should specify a policy of 'affinity'.
- AggregateMultiTenancyIsolation - isolate tenants in specific aggregates. To specify multiple tenants use a comma. Eg. "tenant1,tenant2"
- AggregateImagePropertiesIsolation - isolates hosts based on image properties and aggregate metadata. Use a comma to specify multiple values for the same property. The filter will then ensure at least one value matches.
- MetricsFilter - filters hosts based on metrics weight_setting. Only hosts with the available metrics are passed.
- NUMATopologyFilter - filters hosts based on the NUMA topology requested by the instance, if any.

Now we can focus on these standard filter classes in details. I will pass the simplest ones, such as AllHostsFilter, CoreFilter and RamFilter are, because their functionality is quite simple and can be understood just from the code. For example class RamFilter has the next realization:

```
class RamFilter(filters.BaseHostFilter):
    """Ram Filter with over subscription flag"""

    def host_passes(self, host_state, filter_properties):
        """Only return hosts with sufficient available RAM."""
        instance_type = filter_properties.get('instance_type')
        requested_ram = instance_type['memory_mb']
        free_ram_mb = host_state.free_ram_mb
        total_usable_ram_mb = host_state.total_usable_ram_mb
        used_ram_mb = total_usable_ram_mb - free_ram_mb
        return total_usable_ram_mb * FLAGS.ram_allocation_ratio  - used_ram_mb >= requested_ram
```

Here ram_allocation_ratio means the virtual RAM to physical RAM allocation ratio (it is 1.5 by default). Really, nice and simple.

Next standard filter to describe is AvailabilityZoneFilter and it isn't difficult too. This filter just looks at the availability zone of compute node and availability zone from the properties of the request. Each compute service has its own availability zone. So deployment engineers have an option to run scheduler with availability zones support and can configure availability zones on each compute host. This classes method host_passes returns True if availability zone mentioned in request is the same on the current compute host.

The ImagePropertiesFilter filters hosts based on the architecture, hypervisor type, and virtual machine mode specified in the instance. E.g., an instance might require a host that supports the arm architecture on a qemu compute host. The ImagePropertiesFilter will only pass hosts that can satisfy this request. These instance properties are populated from properties define on the instance's image. E.g. an image can be decorated with these properties using glance image-update img-uuid --property architecture=arm --property hypervisor_type=qemu Only hosts that satisfy these requirements will pass the ImagePropertiesFilter.

ComputeCapabilitiesFilter checks if the host satisfies any extra_specs specified on the instance type.

The extra_specs can contain key/value pairs. The key for the filter is either non-scope format (i.e. no : contained), or scope format in capabilities scope (i.e. capabilities:xxx:yyy). One example of capabilities scope is capabilities:cpu_info:features, which will match host's cpu features capabilities. The ComputeCapabilitiesFilter will only pass hosts whose capabilities satisfy the requested specifications. All hosts are passed if no extra_specs are specified.

ComputeFilter is quite simple and passes any host whose compute service is enabled and operational.

Now we are going to IsolatedHostsFilter. There can be some special hosts reserved for specific images. These hosts are called **isolated**. So the images to run on the isolated hosts are also called isolated. This Scheduler checks if image_isolated flag named in instance specifications is the same that the host has. Isolated hosts can run non isolated images if the flag restrict_isolated_hosts_to_isolated_images is set to false.

DifferentHostFilter - its method host_passes returns True if host to place instance on is different from all the hosts used by set of instances.

SameHostFilter does the opposite to what DifferentHostFilter does. So its host_passes returns True if the host we want to place instance on is one of the set of instances uses.

SimpleCIDRAffinityFilter looks at the subnet mask and investigates if the network address of the current host is in the same sub network as it was defined in the request.

JsonFilter - this filter provides the opportunity to write complicated queries for the hosts capabilities filtering, based on simple JSON-like syntax. There can be used the following operations for the host states properties: =, <, >, in, <=, >=, that can be combined with the following logical operations: not, or, and. For example, there is the query you can find in tests:

```
['and',
    ['>=', '$free_ram_mb', 1024],
    ['>=', '$free_disk_mb', 200 * 1024]
]
```

This query will filter all hosts with free RAM greater or equal than 1024 MB and at the same time with free disk space greater or equal than 200 GB.

Many filters use data from scheduler_hints, that is defined in the moment of creation of the new server for the user. The only exception for this rule is JsonFilter, that takes data from the schedulers HostState data structure directly. Variable naming, such as the $free_ram_mb example above, should be based on those attributes.

The RetryFilter filters hosts that have already been attempted for scheduling. It only passes hosts that have not been previously attempted.

The TrustedFilter filters hosts based on their trust. Only passes hosts that match the trust requested in the extra_specs for the flavor. The key for this filter must be scope format as trust:trusted_host, where trust is the scope of the key and trusted_host is the actual key value. The value of this pair (trusted/untrusted) must match the integrity of a host (obtained from the Attestation service) before it is passed by the TrustedFilter.

The NUMATopologyFilter considers the NUMA topology that was specified for the instance through the use of flavor extra_specs in combination with the image properties, as described in detail in the related nova-spec document:

- http://git.openstack.org/cgit/openstack/nova-specs/tree/specs/juno/virt-driver-numa-placement.rst

and try to match it with the topology exposed by the host, accounting for the ram_allocation_ratio and cpu_allocation_ratio for over-subscription. The filtering is done in the following manner:

- Filter will attempt to pack instance cells onto host cells.
- It will consider the standard over-subscription limits for each host NUMA cell, and provide limits to the compute host accordingly (as mentioned above).
- If instance has no topology defined, it will be considered for any host.

- If instance has a topology defined, it will be considered only for NUMA capable hosts.

To use filters you specify next two settings:

- **`scheduler_available_filters` - Defines filter classes made available to the** scheduler. This setting can be used multiple times.
- `scheduler_default_filters` - Of the available filters, defines those that the scheduler uses by default.

The default values for these settings in nova.conf are:

```
--scheduler_available_filters=nova.scheduler.filters.standard_filters
--scheduler_default_filters=RamFilter,ComputeFilter,AvailabilityZoneFilter,ComputeCapabilitiesFilter,
```

With this configuration, all filters in `nova.scheduler.filters` would be available, and by default the `RamFilter`, `ComputeFilter`, `AvailabilityZoneFilter`, `ComputeCapabilitiesFilter`, `ImagePropertiesFilter`, `ServerGroupAntiAffinityFilter`, and `ServerGroupAffinityFilter` would be used.

If you want to create **your own filter** you just need to inherit from `BaseHostFilter` and implement one method: `host_passes`. This method should return `True` if host passes the filter. It takes `host_state` (describes host) and `filter_properties` dictionary as the parameters.

As an example, nova.conf could contain the following scheduler-related settings:

```
--scheduler_driver=nova.scheduler.FilterScheduler
--scheduler_available_filters=nova.scheduler.filters.standard_filters
--scheduler_available_filters=myfilter.MyFilter
--scheduler_default_filters=RamFilter,ComputeFilter,MyFilter
```

With these settings, nova will use the `FilterScheduler` for the scheduler driver. The standard nova filters and MyFilter are available to the FilterScheduler. The RamFilter, ComputeFilter, and MyFilter are used by default when no filters are specified in the request.

Weights

Filter Scheduler uses the so called **weights** during its work. A weigher is a way to select the best suitable host from a group of valid hosts by giving weights to all the hosts in the list.

In order to prioritize one weigher against another, all the weighers have to define a multiplier that will be applied before computing the weight for a node. All the weights are normalized beforehand so that the multiplier can be applied easily. Therefore the final weight for the object will be:

```
weight = w1_multiplier * norm(w1) + w2_multiplier * norm(w2) + ...
```

A weigher should be a subclass of `weights.BaseHostWeigher` and they must implement the `weight_multiplier` and `weight_object` methods. If the `weight_objects` method is overridden it just return a list of weights, and not modify the weight of the object directly, since final weights are normalized and computed by `weight.BaseWeightHandler`.

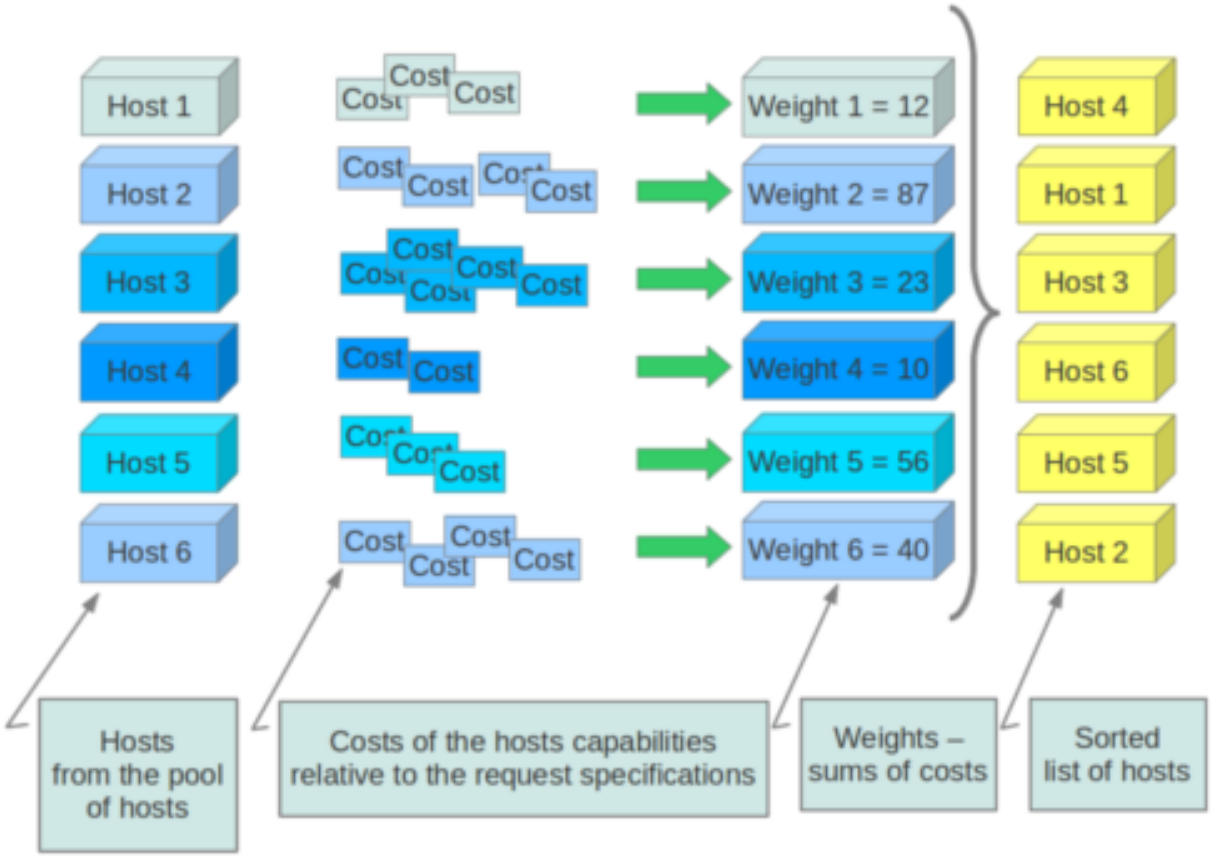
The Filter Scheduler weighs hosts based on the config option *scheduler_weight_classes*, this defaults to *nova.scheduler.weights.all_weighers*, which selects the following weighers:

- `RAMWeigher` Hosts are then weighted and sorted with the largest weight winning. If the multiplier is negative, the host with less RAM available will win (useful for stacking hosts, instead of spreading).
- `MetricsWeigher` This weigher can compute the weight based on the compute node host's various metrics. The to-be weighed metrics and their weighing ratio are specified in the configuration file as the followings:

  ```
  metrics_weight_setting = name1=1.0, name2=-1.0
  ```

- `IoOpsWeigher` The weigher can compute the weight based on the compute node host's workload. The default is to preferably choose light workload compute hosts. If the multiplier is positive, the weigher prefer choosing heavy workload compute hosts, the weighing has the opposite effect of the default.

Filter Scheduler finds local list of acceptable hosts by repeated filtering and weighing. Each time it chooses a host, it virtually consumes resources on it, so subsequent selections can adjust accordingly. It is useful if the customer asks for the some large amount of instances, because weight is computed for each instance requested.



In the end Filter Scheduler sorts selected hosts by their weight and provisions instances on them.

P.S.: you can find more examples of using Filter Scheduler and standard filters in :mod:`nova.tests.scheduler`.

. Copyright (c) 2010 Citrix Systems, Inc. All Rights Reserved.

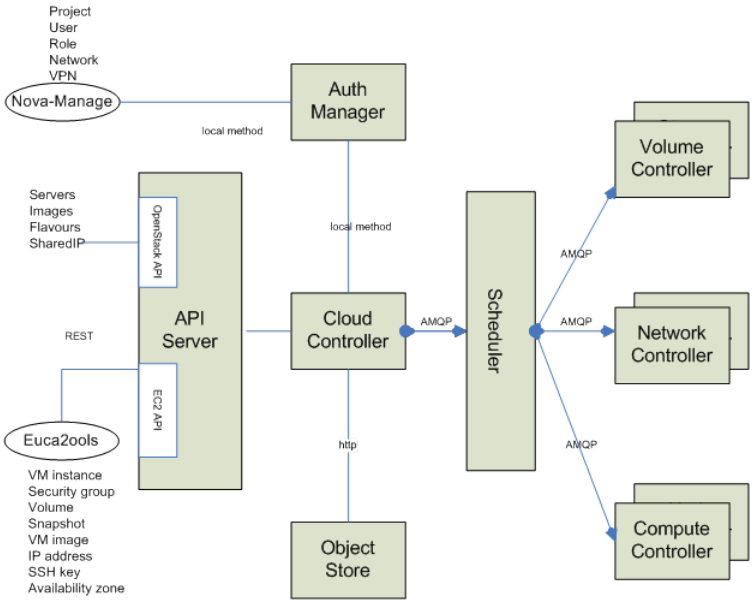
3.3.7 AMQP and Nova

AMQP is the messaging technology chosen by the OpenStack cloud. The AMQP broker, either RabbitMQ or Qpid, sits between any two Nova components and allows them to communicate in a loosely coupled fashion. More precisely,

Nova components (the compute fabric of OpenStack) use Remote Procedure Calls (RPC hereinafter) to communicate to one another; however such a paradigm is built atop the publish/subscribe paradigm so that the following benefits can be achieved:

- Decoupling between client and servant (such as the client does not need to know where the servant's reference is).
- Full a-synchronism between client and servant (such as the client does not need the servant to run at the same time of the remote call).
- Random balancing of remote calls (such as if more servants are up and running, one-way calls are transparently dispatched to the first available servant).

Nova uses direct, fanout, and topic-based exchanges. The architecture looks like the one depicted in the figure below:



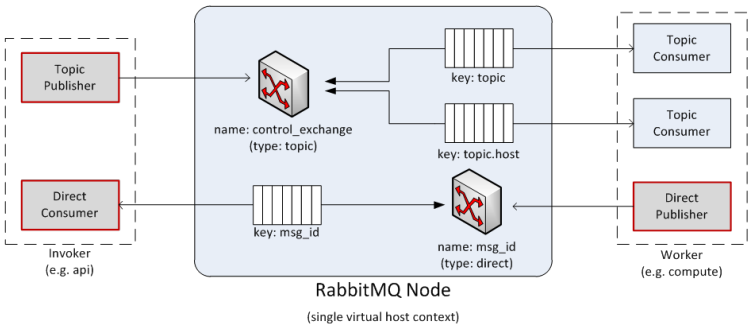
Nova implements RPC (both request+response, and one-way, respectively nicknamed 'rpc.call' and 'rpc.cast') over AMQP by providing an adapter class which take cares of marshaling and unmarshaling of messages into function calls. Each Nova service (for example Compute, Scheduler, etc.) create two queues at the initialization time, one which accepts messages with routing keys 'NODE-TYPE.NODE-ID' (for example compute.hostname) and another, which accepts messages with routing keys as generic 'NODE-TYPE' (for example compute). The former is used specifically when Nova-API needs to redirect commands to a specific node like 'euca-terminate instance'. In this case, only the compute node whose host's hypervisor is running the virtual machine can kill the instance. The API acts as a consumer when RPC calls are request/response, otherwise it acts as a publisher only.

Nova RPC Mappings

The figure below shows the internals of a message broker node (referred to as a RabbitMQ node in the diagrams) when a single instance is deployed and shared in an OpenStack cloud. Every Nova component connects to the message broker and, depending on its personality (for example a compute node or a network node), may use the queue either as an Invoker (such as API or Scheduler) or a Worker (such as Compute or Network). Invokers and Workers do not actually exist in the Nova object model, but we are going to use them as an abstraction for sake of clarity. An Invoker is a component that sends messages in the queuing system via two operations: 1) rpc.call and ii) rpc.cast; a Worker is a component that receives messages from the queuing system and reply accordingly to rpc.call operations.

Figure 2 shows the following internal elements:

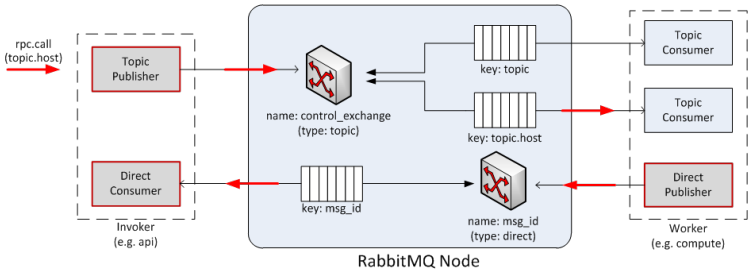
- Topic Publisher: a Topic Publisher comes to life when an rpc.call or an rpc.cast operation is executed; this object is instantiated and used to push a message to the queuing system. Every publisher connects always to the same topic-based exchange; its life-cycle is limited to the message delivery.
- Direct Consumer: a Direct Consumer comes to life if (an only if) a rpc.call operation is executed; this object is instantiated and used to receive a response message from the queuing system; Every consumer connects to a unique direct-based exchange via a unique exclusive queue; its life-cycle is limited to the message delivery; the exchange and queue identifiers are determined by a UUID generator, and are marshaled in the message sent by the Topic Publisher (only rpc.call operations).
- Topic Consumer: a Topic Consumer comes to life as soon as a Worker is instantiated and exists throughout its life-cycle; this object is used to receive messages from the queue and it invokes the appropriate action as defined by the Worker role. A Topic Consumer connects to the same topic-based exchange either via a shared queue or via a unique exclusive queue. Every Worker has two topic consumers, one that is addressed only during rpc.cast operations (and it connects to a shared queue whose exchange key is 'topic') and the other that is addressed only during rpc.call operations (and it connects to a unique queue whose exchange key is 'topic.host').
- Direct Publisher: a Direct Publisher comes to life only during rpc.call operations and it is instantiated to return the message required by the request/response operation. The object connects to a direct-based exchange whose identity is dictated by the incoming message.
- Topic Exchange: The Exchange is a routing table that exists in the context of a virtual host (the multi-tenancy mechanism provided by Qpid or RabbitMQ); its type (such as topic vs. direct) determines the routing policy; a message broker node will have only one topic-based exchange for every topic in Nova.
- Direct Exchange: this is a routing table that is created during rpc.call operations; there are many instances of this kind of exchange throughout the life-cycle of a message broker node, one for each rpc.call invoked.
- Queue Element: A Queue is a message bucket. Messages are kept in the queue until a Consumer (either Topic or Direct Consumer) connects to the queue and fetch it. Queues can be shared or can be exclusive. Queues whose routing key is 'topic' are shared amongst Workers of the same personality.



RPC Calls

The diagram below shows the message flow during an rpc.call operation:

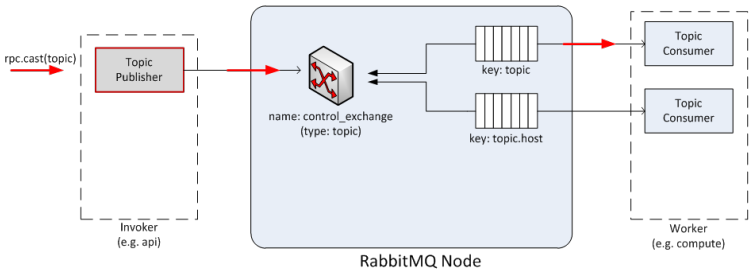
1. a Topic Publisher is instantiated to send the message request to the queuing system; immediately before the publishing operation, a Direct Consumer is instantiated to wait for the response message.
2. once the message is dispatched by the exchange, it is fetched by the Topic Consumer dictated by the routing key (such as 'topic.host') and passed to the Worker in charge of the task.
3. once the task is completed, a Direct Publisher is allocated to send the response message to the queuing system.
4. once the message is dispatched by the exchange, it is fetched by the Direct Consumer dictated by the routing key (such as 'msg_id') and passed to the Invoker.



RPC Casts

The diagram below shows the message flow during an rpc.cast operation:

1. A Topic Publisher is instantiated to send the message request to the queuing system.
2. Once the message is dispatched by the exchange, it is fetched by the Topic Consumer dictated by the routing key (such as 'topic') and passed to the Worker in charge of the task.



AMQP Broker Load

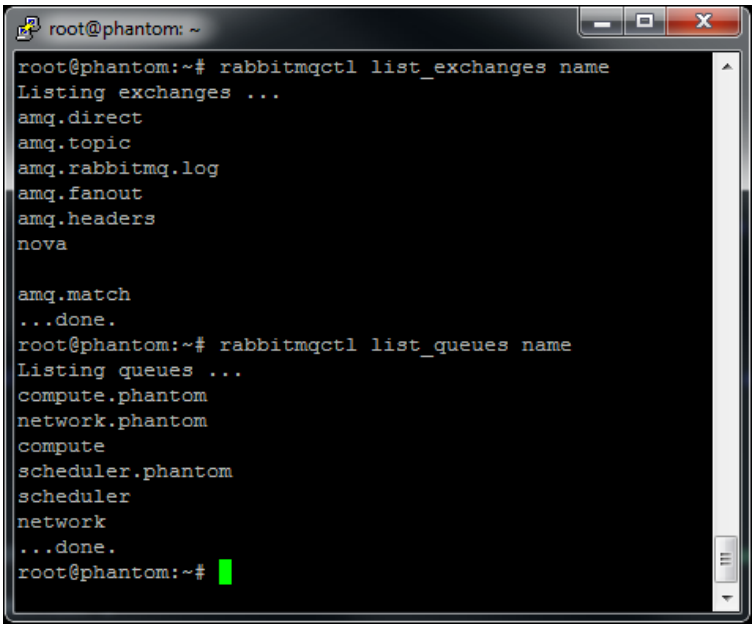
At any given time the load of a message broker node running either Qpid or RabbitMQ is function of the following parameters:

- Throughput of API calls: the number of API calls (more precisely rpc.call ops) being served by the OpenStack cloud dictates the number of direct-based exchanges, related queues and direct consumers connected to them.
- Number of Workers: there is one queue shared amongst workers with the same personality; however there are as many exclusive queues as the number of workers; the number of workers dictates also the number of routing keys within the topic-based exchange, which is shared amongst all workers.

The figure below shows the status of a RabbitMQ node after Nova components' bootstrap in a test environment. Exchanges and queues being created by Nova components are:

- **Exchanges**
    1. nova (topic exchange)
- **Queues**
    1. compute.phantom (phantom is hostname)
    2. compute
    3. network.phantom (phantom is hostname)
    4. network
    5. scheduler.phantom (phantom is hostname)

6. scheduler



RabbitMQ Gotchas

Nova uses Kombu to connect to the RabbitMQ environment. Kombu is a Python library that in turn uses AMQPLib, a library that implements the standard AMQP 0.8 at the time of writing. When using Kombu, Invokers and Workers need the following parameters in order to instantiate a Connection object that connects to the RabbitMQ server (please note that most of the following material can be also found in the Kombu documentation; it has been summarized and revised here for sake of clarity):

- Hostname: The hostname to the AMQP server.
- Userid: A valid username used to authenticate to the server.
- Password: The password used to authenticate to the server.
- Virtual_host: The name of the virtual host to work with. This virtual host must exist on the server, and the user must have access to it. Default is "/".
- Port: The port of the AMQP server. Default is 5672 (amqp).

The following parameters are default:

- Insist: insist on connecting to a server. In a configuration with multiple load-sharing servers, the Insist option tells the server that the client is insisting on a connection to the specified server. Default is False.
- Connect_timeout: the timeout in seconds before the client gives up connecting to the server. The default is no timeout.
- SSL: use SSL to connect to the server. The default is False.

More precisely Consumers need the following parameters:

- Connection: the above mentioned Connection object.
- Queue: name of the queue.
- Exchange: name of the exchange the queue binds to.
- Routing_key: the interpretation of the routing key depends on the value of the exchange_type attribute.

  - Direct exchange: if the routing key property of the message and the routing_key attribute of the queue are identical, then the message is forwarded to the queue.
  - Fanout exchange: messages are forwarded to the queues bound the exchange, even if the binding does not have a key.
  - Topic exchange: if the routing key property of the message matches the routing key of the key according to a primitive pattern matching scheme, then the message is forwarded to the queue. The message routing key then consists of words separated by dots (".", like domain names), and two special characters are available; star ("") and hash ("#"). The star matches any word, and the hash matches zero or more words. For example ".stock.#" matches the routing keys "usd.stock" and "eur.stock.db" but not "stock.nasdaq".

- Durable: this flag determines the durability of both exchanges and queues; durable exchanges and queues remain active when a RabbitMQ server restarts. Non-durable exchanges/queues (transient exchanges/queues) are purged when a server restarts. It is worth noting that AMQP specifies that durable queues cannot bind to transient exchanges. Default is True.
- Auto_delete: if set, the exchange is deleted when all queues have finished using it. Default is False.
- Exclusive: exclusive queues (such as non-shared) may only be consumed from by the current connection. When exclusive is on, this also implies auto_delete. Default is False.
- Exchange_type: AMQP defines several default exchange types (routing algorithms) that covers most of the common messaging use cases.
- Auto_ack: acknowledgement is handled automatically once messages are received. By default auto_ack is set to False, and the receiver is required to manually handle acknowledgment.
- No_ack: it disable acknowledgement on the server-side. This is different from auto_ack in that acknowledgement is turned off altogether. This functionality increases performance but at the cost of reliability. Messages can get lost if a client dies before it can deliver them to the application.
- Auto_declare: if this is True and the exchange name is set, the exchange will be automatically declared at instantiation. Auto declare is on by default. Publishers specify most the parameters of Consumers (such as they do not specify a queue name), but they can also specify the following:
- Delivery_mode: the default delivery mode used for messages. The value is an integer. The following delivery modes are supported by RabbitMQ:
  - 1 or "transient": the message is transient. Which means it is stored in memory only, and is lost if the server dies or restarts.
  - 2 or "persistent": the message is persistent. Which means the message is stored both in-memory, and on disk, and therefore preserved if the server dies or restarts.

The default value is 2 (persistent). During a send operation, Publishers can override the delivery mode of messages so that, for example, transient messages can be sent over a durable queue.

3.3.8 Hooks

Hooks provide a mechanism to extend Nova with custom code through a plugin mechanism.

Named hooks are added to nova code via a decorator that will lazily load plugin code matching the name. The loading works via setuptools entry points.

What are hooks good for?

Hooks are good for anchoring your custom code to Nova internal APIs.

What are hooks NOT good for?

Hooks should not be used when API stability is a key factor. Internal APIs may change. Consider using a notification driver if this is important to you.

Declaring hooks in the Nova codebase

The following example declares a *resize_hook* around the *resize_instance* method:

```
from nova import hooks

@hooks.add_hook("resize_hook")
def resize_instance(self, context, instance, a=1, b=2):
    ...
```

Hook objects can now be attached via entry points to the *resize_hook*.

Adding hook object code

1. Setup a Python package with a setup.py file.
2. Add the following to the setup.py setup call:

   ```
   entry_points = {
       'nova.hooks': [
           'resize_hook=your_package.hooks:YourHookClass',
       ]
   },
   ```

3. *YourHookClass* should be an object with *pre* and/or *post* methods:

   ```
   class YourHookClass(object):

       def pre(self, *args, **kwargs):
           ....

       def post(self, rv, *args, **kwargs):
           ....
   ```

3.3.9 Upgrades

Nova aims to provide upgrades with minimal downtime.

Firstly, the data plane. There should be no VM downtime when you upgrade Nova. Nova has had this since the early days, with the exception of some nova-network related services.

Secondly, we want no downtime during upgrades of the Nova control plane. This document is trying to describe how we can achieve that.

Once we have introduced the key concepts relating to upgrade, we will introduce the process needed for a no downtime upgrade of nova.

Current Database Upgrade Types

Currently Nova has 2 types of database upgrades that are in use.

1. Offline Migrations
2. Online Migrations

Offline Migrations consist of:

1. Database schema migrations from pre-defined migrations in nova/db/sqlalchemy/migrate_repo/versions.
2. *Deprecated* Database data migrations from pre-defined migrations in nova/db/sqlalchemy/migrate_repo/versions.

Online Migrations consist of:

1. Online data migrations from inside Nova object source code.
2. *Future* Online schema migrations using auto-generation from models.

An example of online data migrations are the flavor migrations done as part of Nova object version 1.18. This included a transient migration of flavor storage from one database location to another.

Note: Database downgrades are not supported.

Concepts

Here are the key concepts you need to know before reading the section on the upgrade process:

RPC version pinning Through careful RPC versioning, newer nodes are able to talk to older nova-compute nodes. When upgrading control plane nodes, we can pin them at an older version of the compute RPC API, until all the compute nodes are able to be upgraded. https://wiki.openstack.org/wiki/RpcMajorVersionUpdates

Online Configuration Reload During the upgrade, we pin new serves at the older RPC version. When all services are updated to use newer code, we need to unpin them so we are able to use any new functionality. To avoid having to restart the service, using the current SIGHUP signal handling, or otherwise, ideally we need a way to update the currently running process to use the latest configuration.

Graceful service shutdown Many nova services are python processes listening for messages on a AMQP queue, including nova-compute. When sending the process the SIGTERM the process stops getting new work from its queue, completes any outstanding work, then terminates. During this process, messages can be left on the queue for when the python process starts back up. This gives us a way to shutdown a service using older code, and start up a service using newer code with minimal impact. If its a service that can have multiple workers, like nova-conductor, you can usually add the new workers before the graceful shutdown of the old workers. In the case of singleton services, like nova-compute, some actions could be delayed during the restart, but ideally no actions should fail due to the restart. NOTE: while this is true for the RabbitMQ RPC backend, we need to confirm what happens for other RPC backends.

API load balancer draining When upgrading API nodes, you can make your load balancer only send new connections to the newer API nodes, allowing for a seamless update of your API nodes.

Expand/Contract DB Migrations Modern databases are able to make many schema changes while you are still writing to the database. Taking this a step further, we can make all DB changes by first adding the new structures, expanding. Then you can slowly move all the data into a new location and format. Once that is complete, you can drop bits of the scheme that are no long needed, i.e. contract. We have plans to implement this here: https://review.openstack.org/#/c/102545/5/specs/juno/online-schema-changes.rst,cm

Online Data Migrations using objects In Kilo we are moving all data migration into the DB objects code. When trying to migrate data in the database from the old format to the new format, this is done in the object code when reading or saving things that are in the old format. For records that are not updated, you need to run a background

process to convert those records into the newer format. This process must be completed before you contract the database schema. We have the first example of this happening here: http://specs.openstack.org/openstack/nova-specs/specs/kilo/approved/flavor-from-sysmeta-to-blob.html

DB prune deleted rows Currently resources are soft deleted in the database, so users are able to track instances in the DB that are created and destroyed in production. However, most people have a data retention policy, of say 30 days or 90 days after which they will want to delete those entries. Not deleting those entries affects DB performance as indices grow very large and data migrations take longer as there is more data to migrate.

nova-conductor object backports RPC pinning ensures new services can talk to the older service's method signatures. But many of the parameters are objects that may well be too new for the old service to understand, so you are able to send the object back to the nova-conductor to be downgraded to a version the older service can understand.

Process

NOTE: This still requires much work before it can become reality. This is more an aspirational plan that helps describe how all the pieces of the jigsaw fit together.

This is the planned process for a zero downtime upgrade:

1. Prune deleted DB rows, check previous migrations are complete
2. Expand DB schema (e.g. add new column)
3. Pin RPC versions for all services that are upgraded from this point, using the current version
4. Upgrade all nova-conductor nodes (to do object backports)
5. Upgrade all other services, except nova-compute and nova-api, using graceful shutdown
6. Upgrade nova-compute nodes (this is the bulk of the work).
7. Unpin RPC versions
8. Add new API nodes, and enable new features, while using a load balancer to "drain" the traffic from old API nodes
9. Run the new nova-manage command that ensures all DB records are "upgraded" to new data version
10. "Contract" DB schema (e.g. drop unused columns)

Testing

Once we have all the pieces in place, we hope to move the Grenade testing to follow this new pattern.

The current tests only cover the existing upgrade process where: * old computes can run with new control plane * but control plane is turned off for DB migrations

Unresolved issues

Ideally you could rollback. We would need to add some kind of object data version pinning, so you can be running all new code to some extent, before there is no path back. Or have some way of reversing the data migration before the final contract.

It is unknown how expensive on demand object backports would be. We could instead always send older versions of objects until the RPC pin is removed, but that means we might have new code getting old objects, which is currently not the case.

3.4 Development policies

3.4.1 Blueprints, Specs and Priorities

Like most OpenStack projects, Nova uses blueprints and specifications (specs) to track new features, but not all blueprints require a spec. This document covers when a spec is needed.

Note: Nova's specs live at: specs.openstack.org

Specs

A spec is needed for any feature that requires a design discussion. All features need a blueprint but not all blueprints require a spec.

If a new feature is straightforward enough that it doesn't need any design discussion, then no spec is required. In order to provide the sort of documentation that would otherwise be provided via a spec, the commit message should include a `DocImpact` flag and a thorough description of the feature from a user/operator perspective.

Guidelines for when a feature doesn't need a spec.

- Is the feature a single self contained change?
  - If the feature touches code all over the place, it probably should have a design discussion.
  - If the feature is big enough that it needs more then one commit, it probably should have a design discussion.
- Not an API change.
  - API changes always require a design discussion.

Project Priorities

- Pick several project priority themes, in the form of use cases, to help us prioritize work
  - Generate list of improvement blueprints based on the themes
  - Produce rough draft of list going into summit and finalize the list at the summit
  - Publish list of project priorities and look for volunteers to work on them
- Update spec template to include
  - Specific use cases
  - State if the spec is project priority or not
- Keep an up to date list of project priority blueprints that need code review in an etherpad.
- Consumers of project priority and project priority blueprint lists:
  - Reviewers looking for direction of where to spend their blueprint review time. If a large subset of nova-core doesn't use the project priorities it means the core team is not aligned properly and should revisit the list of project priorities
  - The blueprint approval team, to help find the right balance of blueprints
  - Contributors looking for something to work on
  - People looking for what they can expect in the next release

3.4.2 Development policies

Out Of Tree Support

While nova has many entrypoints and other places in the code that allow for wiring in out of tree code, upstream doesn't actively make any guarantees about these extensibility points; we don't support them, make any guarantees about compatibility, stability, etc.

Public Contractual APIs

Although nova has many internal APIs, they are not all public contractual APIs. Below is a list of our public contractual APIs:

- All REST API

Anything not in this list is considered private, not to be used outside of nova, and should not be considered stable.

REST APIs

Follow the guidelines set in: https://wiki.openstack.org/wiki/APIChangeGuidelines

The canonical source for REST API behavior is the code *not* documentation. Documentation is manually generated after the code by folks looking at the code and writing up what they think it does, and it is very easy to get this wrong.

This policy is in place to prevent us from making backwards incompatible changes to REST APIs.

Patches and Reviews

Merging a patch requires a non-trivial amount of reviewer resources. As a patch author, you should try to offset the reviewer resources spent on your patch by reviewing other patches. If no one does this, the review team (cores and otherwise) become spread too thin.

For review guidelines see: https://wiki.openstack.org/wiki/ReviewChecklist

Reverts for Retrospective Vetos

Sometimes our simple "2 +2s" approval policy will result in errors. These errors might be a bug that was missed, or equally importantly, it might be that other cores feel that there is a need for more discussion on the implementation of a given piece of code.

Rather than an enforced time-based solution - for example, a patch couldn't be merged until it has been up for review for 3 days - we have chosen an honor-based system where core reviewers would not approve potentially contentious patches until the proposal had been sufficiently socialized and everyone had a chance to raise any concerns.

Recognising that mistakes can happen, we also have a policy where contentious patches which were quickly approved should be reverted so that the discussion around the proposal can continue as if the patch had never been merged in the first place. In such a situation, the procedure is:

0. The commit to be reverted must not have been released.
1. The core team member who has a -2 worthy objection should propose a revert, stating the specific concerns that they feel need addressing.
2. Any subsequent patches depending on the to-be-reverted patch may need to be reverted also.

3. Other core team members should quickly approve the revert. No detailed debate should be needed at this point. A -2 vote on a revert is strongly discouraged, because it effectively blocks the right of cores approving the revert from -2 voting on the original patch.
4. The original patch submitter should re-submit the change, with a reference to the original patch and the revert.
5. The original reviewers of the patch should restore their votes and attempt to summarize their previous reasons for their votes.
6. The patch should not be re-approved until the concerns of the people proposing the revert are worked through. A mailing list discussion or design spec might be the best way to achieve this.

3.5 Advanced testing and guides

3.5.1 Guru Meditation Reports

Nova contains a mechanism whereby developers and system administrators can generate a report about the state of a running Nova executable. This report is called a *Guru Meditation Report* (*GMR* for short).

Generating a GMR

A *GMR* can be generated by sending the *USR1* signal to any Nova process with support (see below). The *GMR* will then be outputted standard error for that particular process.

For example, suppose that `nova-api` has process id `8675`, and was run with `2>/var/log/nova/nova-api-err.log`. Then, `kill -USR1 8675` will trigger the Guru Meditation report to be printed to `/var/log/nova/nova-api-err.log`.

Structure of a GMR

The *GMR* is designed to be extensible; any particular executable may add its own sections. However, the base *GMR* consists of several sections:

Package Shows information about the package to which this process belongs, including version information

Threads Shows stack traces and thread ids for each of the threads within this process

Green Threads Shows stack traces for each of the green threads within this process (green threads don't have thread ids)

Configuration Lists all the configuration options currently accessible via the CONF object for the current process

Adding Support for GMRs to New Executables

Adding support for a *GMR* to a given executable is fairly easy.

First import the module (currently residing in oslo-incubator), as well as the Nova version module:

```
from nova.openstack.common.report import guru_meditation_report as gmr
from nova import version
```

Then, register any additional sections (optional):

```
TextGuruMeditation.register_section('Some Special Section',
                                    some_section_generator)
```

Finally (under main), before running the "main loop" of the executable (usually `service.server(server)` or something similar), register the *GMR* hook:

```
TextGuruMeditation.setup_autorun(version)
```

Extending the GMR

As mentioned above, additional sections can be added to the GMR for a particular executable. For more information, see the inline documentation under `nova.openstack.common.report`

3.5.2 Testing NUMA related hardware setup with libvirt

This page describes how to test the libvirt driver's handling of the NUMA placement, large page allocation and CPU pinning features. It relies on setting up a virtual machine as the test environment and requires support for nested virtualization since plain QEMU is not sufficiently functional. The virtual machine will itself be given NUMA topology, so it can then act as a virtual "host" for testing purposes.

Provisioning a virtual machine for testing

The entire test process will take place inside a large virtual machine running Fedora 21. The instructions should work for any other Linux distribution which includes libvirt >= 1.2.9 and QEMU >= 2.1.2

The tests will require support for nested KVM, which is not enabled by default on hypervisor hosts. It must be explicitly turned on in the host when loading the kvm-intel/kvm-amd kernel modules.

On Intel hosts verify it with

```
# cat /sys/module/kvm_intel/parameters/nested
N

# rmmod kvm-intel
# echo "options kvm-intel nested=y" > /etc/modprobe.d/dist.conf
# modprobe kvm-intel

# cat /sys/module/kvm_intel/parameters/nested
Y
```

While on AMD hosts verify it with

```
# cat /sys/module/kvm_amd/parameters/nested
0

# rmmod kvm-amd
# echo "options kvm-amd nested=1" > /etc/modprobe.d/dist.conf
# modprobe kvm-amd

# cat /sys/module/kvm_amd/parameters/nested
1
```

The virt-install command below shows how to provision a basic Fedora 21 x86_64 guest with 8 virtual CPUs, 8 GB of RAM and 20 GB of disk space:

```
# cd /var/lib/libvirt/images
# wget http://download.fedoraproject.org/pub/fedora/linux/releases/test/21-Alpha/Server/x86_64/iso/Fe

# virt-install \
```

```
--name f21x86_64 \
--ram 8000 \
--vcpus 8 \
--file /var/lib/libvirt/images/f21x86_64.img \
--file-size 20
--cdrom /var/lib/libvirt/images/Fedora-Server-netinst-x86_64-21_Alpha.iso \
--os-variant fedora20
```

When the virt-viewer application displays the installer, follow the defaults for the installation with a couple of exceptions

- The automatic disk partition setup can be optionally tweaked to reduce the swap space allocated. No more than 500MB is required, free'ing up an extra 1.5 GB for the root disk.
- Select "Minimal install" when asked for the installation type since a desktop environment is not required.
- When creating a user account be sure to select the option "Make this user administrator" so it gets 'sudo' rights

Once the installation process has completed, the virtual machine will reboot into the final operating system. It is now ready to deploy an OpenStack development environment.

Setting up a devstack environment

For later ease of use, copy your SSH public key into the virtual machine

```
# ssh-copy-id  <IP of VM>
```

Now login to the virtual machine

```
# ssh <IP of VM>
```

We'll install devstack under $HOME/src/cloud/.

```
# mkdir -p $HOME/src/cloud
# cd $HOME/src/cloud
# chmod go+rx $HOME
```

The Fedora minimal install does not contain git and only has the crude & old-fashioned "vi" editor.

```
# sudo yum -y install git emacs
```

At this point a fairly standard devstack setup can be done. The config below is just an example that is convenient to use to place everything in $HOME instead of /opt/stack. Change the IP addresses to something appropriate for your environment of course

```
# git clone git://github.com/openstack-dev/devstack.git
# cd devstack
# cat >>local.conf <<EOF
[[local|localrc]]
DEST=$HOME/src/cloud
DATA_DIR=$DEST/data
SERVICE_DIR=$DEST/status

LOGFILE=$DATA_DIR/logs/stack.log
SCREEN_LOGDIR=$DATA_DIR/logs
VERBOSE=True

disable_service neutron

HOST_IP=192.168.122.50
```

```
FLAT_INTERFACE=eth0
FIXED_RANGE=192.168.128.0/24
FIXED_NETWORK_SIZE=256
FLOATING_RANGE=192.168.129.0/24

MYSQL_PASSWORD=123456
SERVICE_TOKEN=123456
SERVICE_PASSWORD=123456
ADMIN_PASSWORD=123456
RABBIT_PASSWORD=123456

IMAGE_URLS="http://download.cirros-cloud.net/0.3.2/cirros-0.3.2-x86_64-uec.tar.gz"
EOF

# FORCE=yes ./stack.sh
```

Unfortunately while devstack starts various system services and changes various system settings it doesn't make the changes persistent. Fix that now to avoid later surprises after reboots

```
# sudo systemctl enable mysqld.service
# sudo systemctl enable rabbitmq-server.service
# sudo systemctl enable httpd.service

# sudo emacs /etc/sysconfig/selinux
SELINUX=permissive
```

Testing basis non-NUMA usage

First to confirm we've not done anything unusual to the traditional operation of Nova libvirt guests boot a tiny instance

```
# . openrc admin
# nova boot --image cirros-0.3.2-x86_64-uec --flavor m1.tiny cirros1
```

The host will be reporting NUMA topology, but there should only be a single NUMA cell this point.

```
# mysql -u root -p nova
MariaDB [nova]> select numa_topology from compute_nodes;
+--------------------------------------------------------------------------------+
| numa_topology                                                                  |
+--------------------------------------------------------------------------------+
| {
|     "nova_object.name": "NUMATopology",
|     "nova_object.data": {
|         "cells": [{
|                 "nova_object.name": "NUMACell",
|                 "nova_object.data": {
|                     "cpu_usage": 0,
|                     "memory_usage": 0,
|                     "cpuset": [0, 1, 2, 3, 4, 5, 6, 7],
|                     "pinned_cpus": [],
|                     "siblings": [],
|                     "memory": 7793,
|                     "mempages": [
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 987430,
|                                 "used": 0,
```

```
|                                     "size_kb": 4
|                                 },
|                             },
|                             {
|                                 "nova_object.name": "NUMAPagesTopology",
|                                 "nova_object.data": {
|                                     "total": 0,
|                                     "used": 0,
|                                     "size_kb": 2048
|                                 },
|                             }
|                         ],
|                         "id": 0
|                     },
|                 },
|             ]
|         },
| }
+--------------------------------------------------------------------------------+
```

Meanwhile, the guest instance should not have any NUMA configuration recorded

```
MariaDB [nova]> select numa_topology from instance_extra;
+---------------+
| numa_topology |
+---------------+
| NULL          |
+---------------+
```

Reconfiguring the test instance to have NUMA topology

Now that devstack is proved operational, it is time to configure some NUMA topology for the test VM, so that it can be used to verify the OpenStack NUMA support. To do the changes, the VM instance that is running devstack must be shut down.

```
# sudo shutdown -h now
```

And now back on the physical host edit the guest config as root

```
# sudo virsh edit f21x86_64
```

The first thing is to change the <cpu> block to do passthrough of the host CPU. In particular this exposes the "SVM" or "VMX" feature bits to the guest so that "Nested KVM" can work. At the same time we want to define the NUMA topology of the guest. To make things interesting we're going to give the guest an asymmetric topology with 4 CPUS and 4 GBs of RAM in the first NUMA node and 2 CPUs and 2 GB of RAM in the second and third NUMA nodes. So modify the guest XML to include the following CPU XML

```
<cpu mode='host-passthrough'>
  <numa>
    <cell id='0' cpus='0-3' memory='4096000'/>
    <cell id='1' cpus='4-5' memory='2048000'/>
    <cell id='2' cpus='6-7' memory='2048000'/>
  </numa>
</cpu>
```

The guest can now be started again, and ssh back into it

```
 # virsh start f21x86_64

...wait for it to finish booting

 # ssh <IP of VM>
```

Before starting OpenStack services again, it is necessary to reconfigure Nova to enable the NUMA schedular filter. The libvirt virtualization type must also be explicitly set to KVM, so that guests can take advantage of nested KVM.

```
# sudo emacs /etc/nova/nova.conf
```

Set the following parameters:

```
[DEFAULT]
scheduler_default_filters=RetryFilter, AvailabilityZoneFilter, RamFilter, ComputeFilter, ComputeCapab

[libvirt]
virt_type = kvm
```

With that done, OpenStack can be started again

```
# cd $HOME/src/cloud/devstack
# ./rejoin-stack.sh
```

The first thing is to check that the compute node picked up the new NUMA topology setup for the guest

```
# mysql -u root -p nova
MariaDB [nova]> select numa_topology from compute_nodes;
+--------------------------------------------------------------------------------+
| numa_topology                                                                  |
+--------------------------------------------------------------------------------+
| {
|     "nova_object.name": "NUMATopology",
|     "nova_object.data": {
|         "cells": [{
|                 "nova_object.name": "NUMACell",
|                 "nova_object.data": {
|                     "cpu_usage": 0,
|                     "memory_usage": 0,
|                     "cpuset": [0, 1, 2, 3],
|                     "pinned_cpus": [],
|                     "siblings": [],
|                     "memory": 3857,
|                     "mempages": [
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 987430,
|                                 "used": 0,
|                                 "size_kb": 4
|                             },
|                         },
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 0,
|                                 "used": 0,
|                                 "size_kb": 2048
|                             },
```

```
|                         }
|                     ],
|                     "id": 0
|                 },
|             },
|             {
|                 "nova_object.name": "NUMACell",
|                 "nova_object.data": {
|                     "cpu_usage": 0,
|                     "memory_usage": 0,
|                     "cpuset": [4, 5],
|                     "pinned_cpus": [],
|                     "siblings": [],
|                     "memory": 1969,
|                     "mempages": [
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 504216,
|                                 "used": 0,
|                                 "size_kb": 4
|                             },
|                         },
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 0,
|                                 "used": 0,
|                                 "size_kb": 2048
|                             },
|                         }
|                     ],
|                     "id": 1
|                 },
|             },
|             {
|                 "nova_object.name": "NUMACell",
|                 "nova_object.data": {
|                     "cpu_usage": 0,
|                     "memory_usage": 0,
|                     "cpuset": [6, 7],
|                     "pinned_cpus": [],
|                     "siblings": [],
|                     "memory": 1967,
|                     "mempages": [
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 503575,
|                                 "used": 0,
|                                 "size_kb": 4
|                             },
|                         },
|                         {
|                             "nova_object.name": "NUMAPagesTopology",
|                             "nova_object.data": {
|                                 "total": 0,
|                                 "used": 0,
```

```
|                                    "size_kb": 2048
|                                },
|                            }
|                        ],
|                        "id": 2
|                    },
|                }
|            ]
|        },
| }
+--------------------------------------------------------------------------------+
```

This indeed shows that there are now 3 NUMA nodes for the "host" machine, the first with 4 GB of RAM and 4 CPUs, and others with 2 GB of RAM and 2 CPUs each.

Testing instance boot with no NUMA topology requested

For the sake of backwards compatibility, if the NUMA filter is enabled, but the flavor/image does not have any NUMA settings requested, it should be assumed that the guest will have a single NUMA node. The guest should be locked to a single host NUMA node too. Boot a guest with the m1.tiny flavor to test this condition

```
# . openrc admin admin
# nova boot --image cirros-0.3.2-x86_64-uec --flavor m1.tiny cirros1
```

Now look at the libvirt guest XML. It should show that the vCPUs are locked to pCPUs within a particular node.

```
# virsh -c qemu:///system list
....
# virsh -c qemu:///system dumpxml instanceXXXXXX
...
<vcpu placement='static' cpuset='6-7'>1</vcpu>
...
```

This example shows that the guest has been locked to the 3rd NUMA node (which contains pCPUs 6 and 7). Note that there is no explicit NUMA topology listed in the guest XML.

Testing instance boot with 1 NUMA cell requested

Moving forward a little, explicitly tell Nova that the NUMA topology for the guest should have a single NUMA node. This should operate in an identical manner to the default behaviour where no NUMA policy is set. To define the topology we will create a new flavor

```
# nova flavor-create m1.numa 999 1024 1 4
# nova flavor-key m1.numa set hw:numa_nodes=1
# nova flavor-show m1.numa
```

Now boot the guest using this new flavor

```
# nova boot --image cirros-0.3.2-x86_64-uec --flavor m1.numa cirros2
```

Looking at the resulting guest XML from libvirt

```
# virsh -c qemu:///system dumpxml instanceXXXXXX
...
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='0-3'/>
```

```
  <vcpupin vcpu='1' cpuset='0-3'/>
  <vcpupin vcpu='2' cpuset='0-3'/>
  <vcpupin vcpu='3' cpuset='0-3'/>
  <emulatorpin cpuset='0-3'/>
</cputune>
...
<cpu>
  <topology sockets='4' cores='1' threads='1'/>
  <numa>
    <cell id='0' cpus='0-3' memory='1048576'/>
  </numa>
</cpu>
...
<numatune>
  <memory mode='strict' nodeset='0'/>
  <memnode cellid='0' mode='strict' nodeset='0'/>
</numatune>
```

The XML shows:

- Each guest CPU has been pinned to the physical CPUs associated with a particular NUMA node
- The emulator threads have been pinned to the union of all physical CPUs in the host NUMA node that the guest is placed on
- The guest has been given a virtual NUMA topology with a single node holding all RAM and CPUs
- The guest NUMA node has been strictly pinned to a host NUMA node.

As a further sanity test, check what Nova recorded for the instance in the database. This should match the <numatune> information

```
MariaDB [nova]> select numa_topology from instance_extra;
+--------------------------------------------------------------------------------+
| numa_topology                                                                  |
+--------------------------------------------------------------------------------+
| {
|     "nova_object.name": "InstanceNUMATopology",
|     "nova_object.data": {
|         "instance_uuid": "4c2302fe-3f0f-46f1-9f3e-244011f6e03a",
|         "cells": [
|             {
|                 "nova_object.name": "InstanceNUMACell",
|                 "nova_object.data": {
|                     "cpu_topology": null,
|                     "pagesize": null,
|                     "cpuset": [
|                         0,
|                         1,
|                         2,
|                         3
|                     ],
|                     "memory": 1024,
|                     "cpu_pinning_raw": null,
|                     "id": 0
|                 },
|             }
|         ]
|     },
| }
```

```
+--------------------------------------------------------------------------------+
```

Testing instance boot with 2 NUMA cells requested

Now getting more advanced we tell Nova that the guest will have two NUMA nodes. To define the topology we will change the previously defined flavor

```
# nova flavor-key m1.numa set hw:numa_nodes=2
# nova flavor-show m1.numa
```

Now boot the guest using this changed flavor

```
# nova boot --image cirros-0.3.2-x86_64-uec --flavor m1.numa cirros2
```

Looking at the resulting guest XML from libvirt

```
# virsh -c qemu:///system dumpxml instanceXXXXXX
...
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='0-3'/>
  <vcpupin vcpu='1' cpuset='0-3'/>
  <vcpupin vcpu='2' cpuset='4-5'/>
  <vcpupin vcpu='3' cpuset='4-5'/>
  <emulatorpin cpuset='0-5'/>
</cputune>
...
<cpu>
  <topology sockets='4' cores='1' threads='1'/>
  <numa>
    <cell id='0' cpus='0-1' memory='524288'/>
    <cell id='1' cpus='2-3' memory='524288'/>
  </numa>
</cpu>
...
<numatune>
  <memory mode='strict' nodeset='0-1'/>
  <memnode cellid='0' mode='strict' nodeset='0'/>
  <memnode cellid='1' mode='strict' nodeset='1'/>
</numatune>
```

The XML shows:

- Each guest CPU has been pinned to the physical CPUs associated with particular NUMA nodes
- The emulator threads have been pinned to the union of all physical CPUs in the host NUMA nodes that the guest is placed on
- The guest has been given a virtual NUMA topology with two nodes, each holding half the RAM and CPUs
- The guest NUMA nodes have been strictly pinned to different host NUMA node.

As a further sanity test, check what Nova recorded for the instance in the database. This should match the <numatune> information

```
MariaDB [nova]> select numa_topology from instance_extra;
+--------------------------------------------------------------------------------+
| numa_topology                                                                  |
+--------------------------------------------------------------------------------+
| {
```

```
|     "nova_object.name": "InstanceNUMATopology",
|     "nova_object.data": {
|         "instance_uuid": "a14fcd68-567e-4d71-aaa4-a12f23f16d14",
|         "cells": [
|             {
|                 "nova_object.name": "InstanceNUMACell",
|                 "nova_object.data": {
|                     "cpu_topology": null,
|                     "pagesize": null,
|                     "cpuset": [
|                         0,
|                         1
|                     ],
|                     "memory": 512,
|                     "cpu_pinning_raw": null,
|                     "id": 0
|                 },
|             },
|             {
|                 "nova_object.name": "InstanceNUMACell",
|                 "nova_object.data": {
|                     "cpu_topology": null,
|                     "pagesize": null,
|                     "cpuset": [
|                         2,
|                         3
|                     ],
|                     "memory": 512,
|                     "cpu_pinning_raw": null,
|                     "id": 1
|                 },
|             }
|         ]
|     },
| }
|
+--------------------------------------------------------------------------------+
```


3.5.3 Testing Serial Console

The main aim of this feature is exposing an interactive web-based serial consoles through a web-socket proxy. This page describes how to test it from a devstack environement.

Setting up a devstack environment

For instructions on how to setup devstack with serial console support enabled see this guide.

Testing the API

Starting a new instance.

```
# cd devstack && . openrc
# nova boot --flavor 1 --image cirros-0.3.2-x86_64-uec cirros1
```

Nova provides a command *nova get-serial-console* which will returns a URL with a valid token to connect to the serial console of VMs.

```
# nova get-serial-console cirros1
+--------+-----------------------------------------------------------------+
| Type   | Url                                                             |
+--------+-----------------------------------------------------------------+
| serial | ws://127.0.0.1:6083/?token=5f7854b7-bf3a-41eb-857a-43fc33f0b1ec |
+--------+-----------------------------------------------------------------+
```

Currently nova does not provide any client able to connect from an interactive console through a web-socket. A simple client for *test purpose* can be written with few lines of Python.

```
# sudo easy_install ws4py || sudo pip install ws4py
# cat >> client.py <<EOF
import sys
from ws4py.client.threadedclient import WebSocketClient
class LazyClient(WebSocketClient):
    def run(self):
        try:
            while not self.terminated:
                try:
                    b = self.sock.recv(4096)
                    sys.stdout.write(b)
                    sys.stdout.flush()
                except: # socket error expected
                    pass
        finally:
            self.terminate()
if __name__ == '__main__':
    if len(sys.argv) != 2 or not sys.argv[1].startswith("ws"):
        print "Usage %s: Please use websocket url"
        print "Example: ws://127.0.0.1:6083/?token=xxx"
        exit(1)
    try:
        ws = LazyClient(sys.argv[1], protocols=['binary'])
        ws.connect()
        while True:
            # keyboard event...
            c = sys.stdin.read(1)
            if c:
                ws.send(c)
        ws.run_forever()
    except KeyboardInterrupt:
        ws.close()
EOF

# python client.py ws://127.0.0.1:6083/?token=5f7854b7-bf3a-41eb-857a-43fc33f0b1ec
<enter>
cirros1 login
```

3.6 Man Pages

3.6.1 Command-line Utilities

In this section you will find information on Nova's command line utilities.

Reference

nova-all

Server for all Nova services

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-all [options]

DESCRIPTION nova-all is a server daemon that serves all Nova services, each in a separate greenthread

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/api-paste.ini
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-api-ec2

Server for the Nova EC2 API

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-api-ec2 [options]

DESCRIPTION nova-api-ec2 is a server daemon that serves the Nova EC2 API

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/api-paste.ini
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-api-metadata

Server for the Nova Metadata API

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-api-metadata [options]

DESCRIPTION nova-api-metadata is a server daemon that serves the Nova Metadata API

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/api-paste.ini
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-api-os-compute

Server for the Nova OpenStack Compute APIs

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-api-os-compute [options]

DESCRIPTION nova-api-os-compute is a server daemon that serves the Nova OpenStack Compute API

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/api-paste.ini
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-api

Server for the Nova EC2 and OpenStack APIs

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-api [options]

DESCRIPTION nova-api is a server daemon that serves the nova EC2 and OpenStack APIs in separate greenthreads

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/api-paste.ini
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-cert

Server for the Nova Cert

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-cert [options]

DESCRIPTION nova-cert is a server daemon that serves the Nova Cert service for X509 certificates. Used to generate certificates for euca-bundle-image. Only needed for EC2 API.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-compute

Nova Compute Server

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-compute [options]

DESCRIPTION Handles all processes relating to instances (guest vms). nova-compute is responsible for building a disk image, launching it via the underlying virtualization driver, responding to calls to check its state, attaching persistent storage, and terminating it.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-conductor

Server for the Nova Conductor

Author openstack@lists.openstack.org

Date 2012-11-16

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-conductor [options]

DESCRIPTION nova-conductor is a server daemon that serves the Nova Conductor service, which provides coordination and database query support for Nova.

OPTIONS

General options

FILES

- /etc/nova/nova.conf

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-console

Nova Console Server

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-console [options]

DESCRIPTION nova-console is a console Proxy to set up multi-tenant VM console access (i.e. with xvp)

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-consoleauth

Nova Console Authentication Server

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-consoleauth [options]

DESCRIPTION Provides Authentication for nova consoles

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-dhcpbridge

Handles Lease Database updates from DHCP servers

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-dhcpbridge [options]

DESCRIPTION Handles lease database updates from DHCP servers. Used whenever nova is managing DHCP (vlan and flatDHCP). nova-dhcpbridge should not be run as a daemon.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/api-paste.ini
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-manage

control and manage cloud computer instances and images

Author openstack@lists.openstack.org

Date 2012-04-05

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-manage <category> <action> [<args>]

DESCRIPTION nova-manage controls cloud computing instances by managing shell selection, vpn connections, and floating IP address configuration. More information about OpenStack Nova is at http://nova.openstack.org.

OPTIONS The standard pattern for executing a nova-manage command is: `nova-manage <category> <command> [<args>]`

Run without arguments to see a list of available command categories: `nova-manage`

Categories are project, shell, vpn, and floating. Detailed descriptions are below.

You can also run with a category argument such as user to see a list of all commands in that category: `nova-manage floating`

These sections describe the available categories and arguments for nova-manage.

Nova Db `nova-manage db version`

Print the current main database version.

`nova-manage db sync`

Sync the main database up to the most recent version. This is the standard way to create the db as well.

`nova-manage db archive_deleted_rows [--max_rows <number>]`

Move deleted rows from production tables to shadow tables.

`nova-manage db null_instance_uuid_scan [--delete]`

Lists and optionally deletes database records where instance_uuid is NULL.

Nova ApiDb `nova-manage api_db version`

Print the current cells api database version.

`nova-manage api_db sync`

Sync the api cells database up to the most recent version. This is the standard way to create the db as well.

Nova Logs `nova-manage logs errors`

Displays nova errors from log files.

`nova-manage logs syslog <number>`

Displays nova alerts from syslog.

Nova Shell `nova-manage shell bpython`

Starts a new bpython shell.

`nova-manage shell ipython`

Starts a new ipython shell.

`nova-manage shell python`

Starts a new python shell.

`nova-manage shell run`

Starts a new shell using python.

`nova-manage shell script <path/scriptname>`

Runs the named script from the specified path with flags set.

Nova VPN `nova-manage vpn list`

Displays a list of projects, their IP port numbers, and what state they're in.

`nova-manage vpn run <projectname>`

Starts the VPN for the named project.

`nova-manage vpn spawn`

Runs all VPNs.

Nova Floating IPs `nova-manage floating create <ip_range> [--pool <pool>] [--interface <interface>]`

Creates floating IP addresses for the given range, optionally specifying a floating pool and a network interface.

`nova-manage floating delete <ip_range>`

Deletes floating IP addresses in the range given.

`nova-manage floating list`

Displays a list of all floating IP addresses.

Nova Images `nova-manage image image_register <path> <owner>`

Registers an image with the image service.

`nova-manage image kernel_register <path> <owner>`

Registers a kernel with the image service.

`nova-manage image ramdisk_register <path> <owner>`

Registers a ramdisk with the image service.

`nova-manage image all_register <image_path> <kernel_path> <ramdisk_path> <owner>`

Registers an image kernel and ramdisk with the image service.

`nova-manage image convert <directory>`

Converts all images in directory from the old (Bexar) format to the new format.

Nova VM

`nova-manage vm list [host]` Show a list of all instances. Accepts optional hostname (to show only instances on specific host).

`nova-manage live-migration <ec2_id> <destination host name>` Live migrate instance from current host to destination host. Requires instance id (which comes from euca-describe-instance) and destination host name (which can be found from nova-manage service list).

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-network

Nova Network Server

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-network [options]

DESCRIPTION Nova Network is responsible for allocating IPs and setting up the network

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-novncproxy

Websocket novnc Proxy for OpenStack Nova noVNC consoles.

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-novncproxy [options]

DESCRIPTION Websocket proxy that is compatible with OpenStack Nova noVNC consoles.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-objectstore

Nova Objectstore Server

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-objectstore [options]

DESCRIPTION Implementation of an S3-like storage server based on local files.

Useful to test features that will eventually run on S3, or if you want to run something locally that was once running on S3.

We don't support all the features of S3, but it does work with the standard S3 client for the most basic semantics.

Used for testing when do not have OpenStack Swift installed.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-rootwrap

Root wrapper for Nova

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-rootwrap [options]

DESCRIPTION Filters which commands nova is allowed to run as another user.

To use this, you should set the following in nova.conf: rootwrap_config=/etc/nova/rootwrap.conf

You also need to let the nova user run nova-rootwrap as root in sudoers: nova ALL = (root) NOPASSWD: /usr/bin/nova-rootwrap /etc/nova/rootwrap.conf *

To make allowed commands node-specific, your packaging should only install {compute,network}.filters respectively on compute and network nodes (i.e. nova-api nodes should not have any of those files installed).

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-scheduler

Nova Scheduler

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-scheduler [options]

DESCRIPTION Nova Scheduler picks a compute node to run a VM instance.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-spicehtml5proxy

Websocket Proxy for OpenStack Nova SPICE HTML5 consoles.

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-spicehtml5proxy [options]

DESCRIPTION Websocket proxy that is compatible with OpenStack Nova SPICE HTML5 consoles.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-xvpvncproxy

XVP VNC Console Proxy Server

Author openstack@lists.openstack.org

Date 2012-09-27

Copyright OpenStack Foundation

Version 2012.1

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-xvpvncproxy [options]

DESCRIPTION XVP VNC Console Proxy Server

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova bugs are managed at Launchpad Bugs : Nova

nova-serialproxy

Websocket serial Proxy for OpenStack Nova serial ports.

Author openstack@lists.launchpad.net

Date 2014-03-15

Copyright OpenStack Foundation

Version 2014.2

Manual section 1

Manual group cloud computing

SYNOPSIS

nova-serialproxy [options]

DESCRIPTION Websocket proxy that is compatible with OpenStack Nova serial ports.

OPTIONS

General options

FILES

- /etc/nova/nova.conf
- /etc/nova/policy.json
- /etc/nova/rootwrap.conf
- /etc/nova/rootwrap.d/

SEE ALSO

- OpenStack Nova

BUGS

- Nova is sourced in Launchpad so you can view current bugs at OpenStack Nova

3.7 Module Reference

3.7.1 Services, Managers and Drivers

The responsibilities of Services, Managers, and Drivers, can be a bit confusing to people that are new to nova. This document attempts to outline the division of responsibilities to make understanding the system a little bit easier.

Currently, Managers and Drivers are specified by flags and loaded using utils.load_object(). This method allows for them to be implemented as singletons, classes, modules or objects. As long as the path specified by the flag leads to an object (or a callable that returns an object) that responds to getattr, it should work as a manager or driver.

The `nova.service` Module

Generic Node base class for all workers that run on hosts.

class **Service**(*host*, *binary*, *topic*, *manager*, *report_interval=None*, *periodic_enable=None*, *periodic_fuzzy_delay=None*, *periodic_interval_max=None*, *db_allowed=True*, **args*, ***kwargs*)

Bases: `nova.openstack.common.service.Service`

Service object for binaries running on hosts.

A service takes a manager and enables rpc by listening to queues based on topic. It also periodically runs tasks on the manager and reports its state to the database services table.

basic_config_check()

Perform basic config checks before starting processing.

classmethod **create**(*host=None*, *binary=None*, *topic=None*, *manager=None*, *report_interval=None*, *periodic_enable=None*, *periodic_fuzzy_delay=None*, *periodic_interval_max=None*, *db_allowed=True*)

Instantiates class and passes back application object.

Parameters

- **host** – defaults to CONF.host
- **binary** – defaults to basename of executable
- **topic** – defaults to bin_name - 'nova-' part
- **manager** – defaults to CONF.<topic>_manager
- **report_interval** – defaults to CONF.report_interval
- **periodic_enable** – defaults to CONF.periodic_enable
- **periodic_fuzzy_delay** – defaults to CONF.periodic_fuzzy_delay
- **periodic_interval_max** – if set, the max time to wait between runs

kill()

Destroy the service object in the datastore.

periodic_tasks(*raise_on_error=False*)

Tasks to be run at a periodic interval.

start()

stop()

class `WSGIService`(*name*, *loader=None*, *use_ssl=False*, *max_url_len=None*)

Bases: `object`

Provides ability to launch API from a 'paste' configuration.

`reset`()

Reset server greenpool size to default.

Returns None

`start`()

Start serving this service using loaded configuration.

Also, retrieve updated port number in case '0' was passed in, which indicates a random port should be used.

Returns None

`stop`()

Stop serving this API.

Returns None

`wait`()

Wait for the service to stop serving this API.

Returns None

`process_launcher`()

`serve`(*server*, *workers=None*)

`wait`()

The `nova.manager` Module

Base Manager class.

Managers are responsible for a certain aspect of the system. It is a logical grouping of code relating to a portion of the system. In general other components should be using the manager to make changes to the components that it is responsible for.

For example, other components that need to deal with volumes in some way, should do so by calling methods on the VolumeManager instead of directly changing fields in the database. This allows us to keep all of the code relating to volumes in the same place.

We have adopted a basic strategy of Smart managers and dumb data, which means rather than attaching methods to data objects, components should call manager methods that act on the data.

Methods on managers that can be executed locally should be called directly. If a particular method must execute on a remote host, this should be done via rpc to the service that wraps the manager

Managers should be responsible for most of the db access, and non-implementation specific data. Anything implementation specific that can't be generalized should be done by the Driver.

In general, we prefer to have one manager with multiple drivers for different implementations, but sometimes it makes sense to have multiple managers. You can think of it this way: Abstract different overall strategies at the manager level(FlatNetwork vs VlanNetwork), and different implementations at the driver level(LinuxNetDriver vs CiscoNetDriver).

Managers will often provide methods for initial setup of a host or periodic tasks to a wrapping service.

This module provides Manager, a base class for managers.

class **Manager**(*host=None*, *db_driver=None*, *service_name='undefined'*)
Bases: nova.db.base.Base, nova.openstack.common.periodic_task.PeriodicTasks

cleanup_host()
Hook to do cleanup work when the service shuts down.

Child classes should override this method.

init_host()
Hook to do additional manager initialization when one requests the service be started. This is called before any service record is created.

Child classes should override this method.

periodic_tasks(*context*, *raise_on_error=False*)
Tasks to be run at a periodic interval.

post_start_hook()
Hook to provide the manager the ability to do additional start-up work immediately after a service creates RPC consumers and starts 'running'.

Child classes should override this method.

pre_start_hook()
Hook to provide the manager the ability to do additional start-up work before any RPC queues/consumers are created. This is called after other initialization has succeeded and a service record is created.

Child classes should override this method.

Implementation-Specific Drivers

A manager will generally load a driver for some of its tasks. The driver is responsible for specific implementation details. Anything running shell commands on a host, or dealing with other non-python code should probably be happening in a driver.

Drivers should minimize touching the database, although it is currently acceptable for implementation specific data. This may be reconsidered at some point.

It usually makes sense to define an Abstract Base Class for the specific driver (i.e. VolumeDriver), to define the methods that a different driver would need to implement.

3.7.2 The nova.api.auth Module

Common Auth Middleware.

class **InjectContext**(*context*, **args*, ***kwargs*)
Bases: nova.wsgi.Middleware

Add a 'nova.context' to WSGI environ.

class **NovaKeystoneContext**(*application*)
Bases: nova.wsgi.Middleware

Make a request context from keystone headers.

pipeline_factory(*loader*, *global_conf*, ***local_conf*)
A paste pipeline replica that keys off of auth_strategy.

pipeline_factory_v21(*loader*, *global_conf*, ***local_conf*)
A paste pipeline replica that keys off of auth_strategy.

pipeline_factory_v3 (*loader*, *global_conf*, ***local_conf*)
A paste pipeline replica that keys off of auth_strategy.

3.7.3 The nova.api.compute_req_id Module

Middleware that ensures x-compute-request-id

Using this middleware provides a convenient way to attach the x-compute-request-id to only v2 responses. Previously, this header was set in api/openstack/wsgi.py

Responses for APIv3 are taken care of by the request_id middleware provided in oslo.

class ComputeReqIdMiddleware (*application*)
Bases: oslo_middleware.base.Middleware

3.7.4 The nova.api.ec2.apirequest Module

APIRequest class

class APIRequest (*controller*, *action*, *version*, *args*)
Bases: object

invoke (*context*)

3.7.5 The nova.api.ec2.cloud Module

Cloud Controller: Implementation of EC2 REST API calls, which are dispatched to other nodes via AMQP RPC. State is via distributed datastore.

class CloudController
Bases: object

CloudController provides the critical dispatch between inbound API calls through the endpoint and messages sent to the other nodes.

allocate_address (*context*, ***kwargs*)

associate_address (*context*, *instance_id*, *public_ip*, ***kwargs*)

attach_volume (*context*, *volume_id*, *instance_id*, *device*, ***kwargs*)

authorize_security_group_ingress (*context*, *group_name=None*, *group_id=None*, ***kwargs*)

create_image (*context*, *instance_id*, ***kwargs*)

create_key_pair (*context*, *key_name*, ***kwargs*)

create_security_group (*context*, *group_name*, *group_description*)

create_snapshot (*context*, *volume_id*, ***kwargs*)

create_tags (*context*, ***kwargs*)
Add tags to a resource

Returns True on success, error on failure.

Parameters **context** – context under which the method is called

create_volume (*context*, ***kwargs*)

delete_key_pair(*context*, *key_name*, ***kwargs*)

delete_security_group(*context*, *group_name=None*, *group_id=None*, ***kwargs*)

delete_snapshot(*context*, *snapshot_id*, ***kwargs*)

delete_tags(*context*, ***kwargs*)
Delete tags

Returns True on success, error on failure.

Parameters **context** – context under which the method is called

delete_volume(*context*, *volume_id*, ***kwargs*)

deregister_image(*context*, *image_id*, ***kwargs*)

describe_addresses(*context*, *public_ip=None*, ***kwargs*)

describe_availability_zones(*context*, ***kwargs*)

describe_image_attribute(*context*, *image_id*, *attribute*, ***kwargs*)

describe_images(*context*, *image_id=None*, ***kwargs*)

describe_instance_attribute(*context*, *instance_id*, *attribute*, ***kwargs*)

describe_instances(*context*, ***kwargs*)

describe_instances_v6(*context*, ***kwargs*)

describe_key_pairs(*context*, *key_name=None*, ***kwargs*)

describe_regions(*context*, *region_name=None*, ***kwargs*)

describe_security_groups(*context*, *group_name=None*, *group_id=None*, ***kwargs*)

describe_snapshots(*context*, *snapshot_id=None*, *owner=None*, *restorable_by=None*, ***kwargs*)

describe_tags(*context*, ***kwargs*)
List tags

Returns a dict with a single key 'tagSet' on success, error on failure.

Parameters **context** – context under which the method is called

describe_volumes(*context*, *volume_id=None*, ***kwargs*)

detach_volume(*context*, *volume_id*, ***kwargs*)

disassociate_address(*context*, *public_ip*, ***kwargs*)

get_console_output(*context*, *instance_id*, ***kwargs*)

get_password_data(*context*, *instance_id*, ***kwargs*)

import_key_pair(*context*, *key_name*, *public_key_material*, ***kwargs*)

modify_image_attribute(*context*, *image_id*, *attribute*, *operation_type*, ***kwargs*)

reboot_instances(*context*, *instance_id*, ***kwargs*)
instance_id is a list of instance ids.

register_image(*context*, *image_location=None*, ***kwargs*)

release_address(*context*, *public_ip*, ***kwargs*)

revoke_security_group_ingress(*context*, *group_name=None*, *group_id=None*, ***kwargs*)

run_instances(*context*, ***kwargs*)

start_instances(*context*, *instance_id*, ***kwargs*)
Start each instances in instance_id. Here instance_id is a list of instance ids

stop_instances(*context*, *instance_id*, ***kwargs*)
Stop each instances in instance_id. Here instance_id is a list of instance ids

terminate_instances(*context*, *instance_id*, ***kwargs*)
Terminate each instance in instance_id, which is a list of ec2 ids. instance_id is a kwarg so its name cannot be modified.

update_image(*context*, *image_id*, ***kwargs*)

class CloudSecurityGroupNeutronAPI(*skip_policy_check=False*)
Bases: nova.api.ec2.cloud.EC2SecurityGroupExceptions, nova.network.security_group.neutron_driver.SecurityGroupAPI

class CloudSecurityGroupNovaAPI(*skip_policy_check=False*, ***kwargs*)
Bases: nova.api.ec2.cloud.EC2SecurityGroupExceptions, nova.compute.api.SecurityGroupAPI

class EC2SecurityGroupExceptions
Bases: object

static raise_group_already_exists(*msg*)

static raise_invalid_cidr(*cidr*, *decoding_exception=None*)

static raise_invalid_group(*msg*)

static raise_invalid_property(*msg*)

static raise_not_found(*msg*)

static raise_over_quota(*msg*)

get_cloud_security_group_api()

validate_instance_id(*instance_id*)

validate_volume_id(*volume_id*)

3.7.6 The nova.api.ec2.ec2utils Module

camelcase_to_underscore(*str*)

dict_from_dotted_str(*items*)
parse multi dot-separated argument into dict. EBS boot uses multi dot-separated arguments like BlockDeviceMapping.1.DeviceName=snap-id Convert the above into {'block_device_mapping': {'1': {'device_name': snap-id}}}

ec2_id_to_glance_id(*context*, *ec2_id*)

ec2_id_to_id(*ec2_id*)
Convert an ec2 ID (i-[base 16 number]) to an instance id (int).

ec2_inst_id_to_uuid(*context*, *ec2_id*)
"Convert an instance id to uuid.

ec2_snap_id_to_uuid(*ec2_id*)
Get the corresponding UUID for the given ec2-id.

ec2_vol_id_to_uuid(*ec2_id*)
Get the corresponding UUID for the given ec2-id.

get_instance_uuid_from_int_id(*context*, *reqid*)

get_int_id_from_instance_uuid(*context*, *reqid*)

get_int_id_from_snapshot_uuid(*context*, *reqid*)

get_int_id_from_volume_uuid(*context*, *reqid*)

get_ip_info_for_instance(*context*, *instance*)
Return a dictionary of IP information for an instance.

get_ip_info_for_instance_from_nw_info(*nw_info*)

get_snapshot_uuid_from_int_id(*context*, *reqid*)

get_volume_uuid_from_int_id(*context*, *reqid*)

glance_id_to_ec2_id(*context*, *glance_id*, *image_type='ami'*)

glance_id_to_id(*context*, *reqid*)
Convert a glance id to an internal (db) id.

id_to_ec2_id(*instance_id*, *template='i-%08x'*)
Convert an instance ID (int) to an ec2 ID (i-[base 16 number]).

id_to_ec2_inst_id(*instance_id*)
Get or create an ec2 instance ID (i-[base 16 number]) from uuid.

id_to_ec2_snap_id(*snapshot_id*)
Get or create an ec2 volume ID (vol-[base 16 number]) from uuid.

id_to_ec2_vol_id(*volume_id*)
Get or create an ec2 volume ID (vol-[base 16 number]) from uuid.

id_to_glance_id(*context*, *reqid*)
Convert an internal (db) id to a glance id.

image_ec2_id(*image_id*, *image_type='ami'*)
Returns image ec2_id using id and three letter type.

image_type(*image_type*)
Converts to a three letter image type.

aki, kernel => aki ari, ramdisk => ari anything else => ami

is_ec2_timestamp_expired(*request*, *expires=None*)
Checks the timestamp or expiry time included in an EC2 request and returns true if the request is expired

memoize(*func*)

regex_from_ec2_regex(*ec2_re*)
Converts an EC2-style regex to a python regex. Approach is based on python fnmatch.

reset_cache()

resource_type_from_id(*context*, *resource_id*)
Get resource type by ID

Returns a string representation of the Amazon resource type, if known. Returns None on failure.

Parameters

- **context** – context under which the method is called
- **resource_id** – resource_id to evaluate

search_opts_from_filters(*filters*)

status_to_ec2_attach_status(*volume*)
Get the corresponding EC2 attachment state.

According to EC2 API, the valid attachment status in response is: attaching | attached | detaching | detached

3.7.7 The nova.api.ec2.faults Module

exception Fault(*exception*)
Bases: webob.exc.HTTPException

Captures exception and return REST Response.

ec2_error_response(*request_id*, *code*, *message*, *status=500*)
Helper to construct an EC2 compatible error response.

3.7.8 The nova.api.ec2.inst_state Module

name_to_code(*name*)

3.7.9 The nova.api.manager Module

class MetadataManager(**args*, ***kwargs*)
Bases: nova.manager.Manager

Metadata Manager.

This class manages the Metadata API service initialization. Currently, it just adds an iptables filter rule for the metadata service.

3.7.10 The nova.api.metadata.base Module

Instance Metadata information.

class InstanceMetadata(*instance*, *address=None*, *content=None*, *extra_md=None*, *network_info=None*, *vd_driver=None*, *network_metadata=None*)
Bases: object

Instance metadata.

get_ec2_item(*path_tokens*)

get_ec2_metadata(*version*)

get_mimetype()

get_openstack_item(*path_tokens*)

lookup(*path*)

metadata_for_config_drive()
Yields (path, value) tuples for metadata elements.

set_mimetype(*mime_type*)

exception InvalidMetadataPath
Bases: exceptions.Exception

exception InvalidMetadataVersion
Bases: exceptions.Exception

class **RouteConfiguration**(*path_handler*)
Bases: object

Routes metadata paths to request handlers.

handle_path(*path_tokens*)

class **VendorDataDriver**(**args*, ***kwargs*)
Bases: object

The base VendorData Drivers should inherit from.

get()
Return a dictionary of primitives to be rendered in metadata

Returns A dictionary or primitives.

ec2_md_print(*data*)

find_path_in_tree(*data*, *path_tokens*)

get_metadata_by_address(*address*)

get_metadata_by_instance_id(*instance_id*, *address*, *ctxt=None*)

3.7.11 The nova.api.metadata.handler Module

Metadata request handler.

class **MetadataRequestHandler**
Bases: nova.wsgi.Application

Serve metadata.

get_metadata_by_instance_id(*instance_id*, *address*)

get_metadata_by_remote_address(*address*)

3.7.12 The nova.api.metadata.password Module

convert_password(*context*, *password*)
Stores password as system_metadata items.

Password is stored with the keys 'password_0' -> 'password_3'.

extract_password(*instance*)

handle_password(*req*, *meta_data*)

3.7.13 The nova.api.metadata.vendordata_json Module

Render Vendordata as stored in configured file.

class **JsonFileVendorData**(**args*, ***kwargs*)
Bases: nova.api.metadata.base.VendorDataDriver

get()

3.7.14 The nova.api.openstack.api_version_request Module

class APIVersionRequest(*version_string=None*)
Bases: object

This class represents an API Version Request with convenience methods for manipulation and comparison of version numbers that we need to do to implement microversions.

get_string()
Converts object to string representation which if used to create an APIVersionRequest object results in the same version request.

is_null()

matches(*min_version*, *max_version*)
Returns whether the version object represents a version greater than or equal to the minimum version and less than or equal to the maximum version.

@param min_version: Minimum acceptable version. @param max_version: Maximum acceptable version. @returns: boolean

If min_version is null then there is no minimum limit. If max_version is null then there is no maximum limit. If self is null then raise ValueError

max_api_version()

min_api_version()

3.7.15 The nova.api.openstack.auth Module

class NoAuthMiddleware(*application*)
Bases: nova.api.openstack.auth.NoAuthMiddlewareBase

Return a fake token if one isn't specified.

noauth2 is a variation on noauth that only provides admin privs if 'admin' is provided as the user id. We will deprecate the NoAuthMiddlewareOld for future removal so we don't need to maintain both code paths.

class NoAuthMiddlewareBase(*application*)
Bases: nova.wsgi.Middleware

Return a fake token if one isn't specified.

base_call(*req*, *project_id_in_path*, *always_admin=True*)

class NoAuthMiddlewareOld(*application*)
Bases: nova.api.openstack.auth.NoAuthMiddlewareBase

Return a fake token if one isn't specified.

This is the Deprecated version of noauth, and should be removed in the Liberty cycle.

class NoAuthMiddlewareV3(*application*)
Bases: nova.api.openstack.auth.NoAuthMiddlewareBase

Return a fake token if one isn't specified.

3.7.16 The nova.api.openstack.common Module

class ViewBuilder
Bases: object

Model API responses as dictionaries.

check_cells_enabled(*function*)

check_img_metadata_properties_quota(*context*, *metadata*)

check_snapshots_enabled(*f*)

dict_to_query_str(*params*)

get_flavor(*context*, *flavor_id*)

get_id_from_href(*href*)
Return the id or uuid portion of a url.

Given: 'http://www.foo.com/bar/123?q=4' Returns: '123'

Given: 'http://www.foo.com/bar/abc123?q=4' Returns: 'abc123'

get_instance(*compute_api*, *context*, *instance_id*, *expected_attrs=None*)
Fetch an instance from the compute API, handling error checking.

get_limit_and_marker(*request*, *max_limit=1000*)
get limited parameter from request.

get_networks_for_instance(*context*, *instance*)
Returns a prepared nw_info list for passing into the view builders

We end up with a data structure like:

```
{'public': {'ips': [{'address': '10.0.0.1',
                     'version': 4,
                     'mac_address': 'aa:aa:aa:aa:aa:aa'},
                    {'address': '2001::1',
                     'version': 6,
                     'mac_address': 'aa:aa:aa:aa:aa:aa'}],
            'floating_ips': [{'address': '172.16.0.1',
                              'version': 4,
                              'mac_address': 'aa:aa:aa:aa:aa:aa'},
                             {'address': '172.16.2.1',
                              'version': 4,
                              'mac_address': 'aa:aa:aa:aa:aa:aa'}]},
 ...}
```

get_networks_for_instance_from_nw_info(*nw_info*)

get_pagination_params(*request*)
Return marker, limit tuple from request.

Parameters request – *wsgi.Request* possibly containing 'marker' and 'limit' GET variables. 'marker' is the id of the last element the client has seen, and 'limit' is the maximum number of items to return. If 'limit' is not specified, 0, or > max_limit, we default to max_limit. Negative values for either marker or limit will cause exc.HTTPBadRequest() exceptions to be raised.

get_sort_params(*input_params*, *default_key='created_at'*, *default_dir='desc'*)
Retrieves sort keys/directions parameters.

Processes the parameters to create a list of sort keys and sort directions that correspond to the 'sort_key' and 'sort_dir' parameter values. These sorting parameters can be specified multiple times in order to generate the list of sort keys and directions.

The input parameters are not modified.

Parameters

- **input_params** – webob.multidict of request parameters (from nova.wsgi.Request.params)
- **default_key** – default sort key value, added to the list if no 'sort_key' parameters are supplied
- **default_dir** – default sort dir value, added to the list if no 'sort_dir' parameters are supplied

Returns list of sort keys, list of sort dirs

limited(*items*, *request*, *max_limit=1000*)
Return a slice of items according to requested offset and limit.

Parameters

- **items** – A sliceable entity
- **request** – `wsgi.Request` possibly containing 'offset' and 'limit' GET variables. 'offset' is where to start in the list, and 'limit' is the maximum number of items to return. If 'limit' is not specified, 0, or > max_limit, we default to max_limit. Negative values for either offset or limit will cause exc.HTTPBadRequest() exceptions to be raised.
- **max_limit** – The maximum number of items to return from 'items'

raise_feature_not_supported(*msg=None*)

raise_http_conflict_for_instance_invalid_state(*exc*, *action*, *server_id*)
Raises a webob.exc.HTTPConflict instance containing a message appropriate to return via the API based on the original InstanceInvalidState exception.

remove_version_from_href(*href*)
Removes the first api version from the href.

Given: 'http://www.nova.com/v1.1/123' Returns: 'http://www.nova.com/123'

Given: 'http://www.nova.com/v1.1' Returns: 'http://www.nova.com'

status_from_state(*vm_state*, *task_state='default'*)
Given vm_state and task_state, return a status string.

task_and_vm_state_from_status(*statuses*)
Map the server's multiple status strings to list of vm states and list of task states.

3.7.17 The `nova.api.openstack.compute.consoles` Module

class Controller
Bases: `object`

The Consoles controller for the OpenStack API.

create(*req*, *server_id*, *body*)
Creates a new console.

delete(*req*, *server_id*, *id*)
Deletes a console.

index(*req*, *server_id*)
Returns a list of consoles for this instance.

show(*req*, *server_id*, *id*)
Shows in-depth information on a specific console.

create_resource()

3.7.18 The nova.api.openstack.compute.contrib.admin_actions Module

class AdminActionsController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'suspend': '_suspend', 'injectNetworkInfo': '_inject_network_info', 'resume': '_resume', 'migrate':

wsgi_extensions = []

class Admin_actions(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Enable admin-only server actions

Actions include: pause, unpause, suspend, resume, migrate, resetNetwork, injectNetworkInfo, lock, unlock, createBackup

alias = 'os-admin-actions'

get_controller_extensions()

name = 'AdminActions'

namespace = 'http://docs.openstack.org/compute/ext/admin-actions/api/v1.1'

updated = '2011-09-20T00:00:00Z'

authorize(*context*, *action_name*)

3.7.19 The nova.api.openstack.compute.contrib.agents Module

class AgentController
Bases: object

The agent is talking about guest agent.The host can use this for things like accessing files on the disk, configuring networking, or running other applications/scripts in the guest while it is running. Typically this uses some hypervisor-specific transport to avoid being dependent on a working network configuration. Xen, VMware, and VirtualBox have guest agents,although the Xen driver is the only one with an implementation for managing them in openstack. KVM doesn't really have a concept of a guest agent (although one could be written).

You can find the design of agent update in this link: http://wiki.openstack.org/AgentUpdate and find the code in nova.virt.xenapi.vmops.VMOps._boot_new_instance. In this design We need update agent in guest from host, so we need some interfaces to update the agent info in host.

You can find more information about the design of the GuestAgent in the following link: http://wiki.openstack.org/GuestAgent http://wiki.openstack.org/GuestAgentXenStoreCommunication

create(*req*, *body*)
Creates a new agent build.

delete(*req*, *id*)
Deletes an existing agent build.

index(*req*)
Return a list of all agent builds. Filter by hypervisor.

update(*req*, *id*, *body*)
Update an existing agent build.

class Agents(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Agents support.

alias = 'os-agents'

get_resources()

name = 'Agents'

namespace = 'http://docs.openstack.org/compute/ext/agents/api/v2'

updated = '2012-10-28T00:00:00Z'

3.7.20 The nova.api.openstack.compute.contrib.aggregates Module

The Aggregate admin API extension.

class AggregateController

Bases: object

The Host Aggregates API controller for the OpenStack API.

action(*req*, *id*, *body*)

create(*req*, *body*)

Creates an aggregate, given its name and optional availability zone.

delete(*req*, *id*)

Removes an aggregate by id.

index(*req*)

Returns a list a host aggregate's id, name, availability_zone.

show(*req*, *id*)

Shows the details of an aggregate, hosts and metadata included.

update(*req*, *id*, *body*)

Updates the name and/or availability_zone of given aggregate.

class Aggregates(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Admin-only aggregate administration.

alias = 'os-aggregates'

get_resources()

name = 'Aggregates'

namespace = 'http://docs.openstack.org/compute/ext/aggregates/api/v1.1'

updated = '2012-01-12T00:00:00Z'

get_host_from_body(*fn*)

Makes sure that the host exists.

3.7.21 The nova.api.openstack.compute.contrib.assisted_volume_snapshots Module

class AssistedVolumeSnapshotsController

Bases: nova.api.openstack.wsgi.Controller

create(*req*, *body*)

Creates a new snapshot.

delete(*req*, *id*)
Delete a snapshot.

wsgi_actions = {}

wsgi_extensions = []

class **Assisted_volume_snapshots**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Assisted volume snapshots.

alias = 'os-assisted-volume-snapshots'

get_resources()

name = 'AssistedVolumeSnapshots'

namespace = 'http://docs.openstack.org/compute/ext/assisted-volume-snapshots/api/v2'

updated = '2013-08-29T00:00:00Z'

3.7.22 The nova.api.openstack.compute.contrib.attach_interfaces Module

The instance interfaces extension.

class **Attach_interfaces**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Attach interface support.

alias = 'os-attach-interfaces'

get_resources()

name = 'AttachInterfaces'

namespace = 'http://docs.openstack.org/compute/ext/interfaces/api/v1.1'

updated = '2012-07-22T00:00:00Z'

class **InterfaceAttachmentController**
Bases: object

The interface attachment API controller for the OpenStack API.

create(*req*, *server_id*, *body*)
Attach an interface to an instance.

delete(*req*, *server_id*, *id*)
Detach an interface from an instance.

index(*req*, *server_id*)
Returns the list of interface attachments for a given instance.

show(*req*, *server_id*, *id*)
Return data about the given interface attachment.

3.7.23 The nova.api.openstack.compute.contrib.availability_zone Module

class AvailabilityZoneController
Bases: nova.api.openstack.wsgi.Controller

The Availability Zone API controller for the OpenStack API.

detail (*req*)
Returns a detailed list of availability zone.

index (*req*)
Returns a summary list of availability zone.

wsgi_actions = {}

wsgi_extensions = []

class Availability_zone (*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

1.Add availability_zone to the Create Server v1.1 API.

2.Add availability zones describing.

alias = 'os-availability-zone'

get_resources ()

name = 'AvailabilityZone'

namespace = 'http://docs.openstack.org/compute/ext/availabilityzone/api/v1.1'

updated = '2012-12-21T00:00:00Z'

3.7.24 The nova.api.openstack.compute.contrib.baremetal_ext_status Module

class Baremetal_ext_status (*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Add extended status in Baremetal Nodes v2 API.

alias = 'os-baremetal-ext-status'

name = 'BareMetalExtStatus'

namespace = 'http://docs.openstack.org/compute/ext/baremetal_ext_status/api/v2'

updated = '2013-08-27T00:00:00Z'

3.7.25 The nova.api.openstack.compute.contrib.baremetal_nodes Module

The bare-metal admin extension with Ironic Proxy.

class BareMetalNodeController (*ext_mgr=None*, **args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

The Bare-Metal Node API controller for the OpenStack API.

Ironic is used for the following commands: 'baremetal-node-list' 'baremetal-node-show'

create (*req*, *body*)

delete(*req*, *id*)

index(*req*)

show(*req*, *id*)

wsgi_actions = {'add_interface': '_add_interface', 'remove_interface': '_remove_interface'}

wsgi_extensions = []

class **Baremetal_nodes**(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Admin-only bare-metal node administration.

alias = 'os-baremetal-nodes'

get_resources()

name = 'BareMetalNodes'

namespace = 'http://docs.openstack.org/compute/ext/baremetal_nodes/api/v2'

updated = '2013-01-04T00:00:00Z'

3.7.26 The nova.api.openstack.compute.contrib.block_device_mapping_v2_boot Module

class **Block_device_mapping_v2_boot**(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Allow boot with the new BDM data format.

alias = 'os-block-device-mapping-v2-boot'

name = 'BlockDeviceMappingV2Boot'

namespace = 'http://docs.openstack.org/compute/ext/block_device_mapping_v2_boot/api/v2'

updated = '2013-07-08T00:00:00Z'

3.7.27 The nova.api.openstack.compute.contrib.cell_capacities Module

class **Cell_capacities**(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adding functionality to get cell capacities.

alias = 'os-cell-capacities'

name = 'CellCapacities'

namespace = 'http://docs.openstack.org/compute/ext/cell_capacities/api/v1.1'

updated = '2013-05-27T00:00:00Z'

3.7.28 The nova.api.openstack.compute.contrib.cells Module

The cells extension.

class Cells(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Enables cells-related functionality such as adding neighbor cells, listing neighbor cells, and getting the capabilities of the local cell.

alias = 'os-cells'

get_resources()

name = 'Cells'

namespace = 'http://docs.openstack.org/compute/ext/cells/api/v1.1'

updated = '2013-05-14T00:00:00Z'

class Controller(*ext_mgr*)
Bases: object

Controller for Cell resources.

capacities(**args*, ***kwargs*)
Return capacities for a given cell or all cells.

create(**args*, ***kwargs*)
Create a child cell entry.

delete(**args*, ***kwargs*)
Delete a child or parent cell entry. 'id' is a cell name.

detail(**args*, ***kwargs*)
Return all cells in detail.

index(**args*, ***kwargs*)
Return all cells in brief.

info(**args*, ***kwargs*)
Return name and capabilities for this cell.

show(**args*, ***kwargs*)
Return data about the given cell name. 'id' is a cell name.

sync_instances(**args*, ***kwargs*)
Tell all cells to sync instance info.

update(**args*, ***kwargs*)
Update a child cell entry. 'id' is the cell name to update.

3.7.29 The nova.api.openstack.compute.contrib.certificates Module

class Certificates(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Certificates support.

alias = 'os-certificates'

get_resources()

name = 'Certificates'

namespace = 'http://docs.openstack.org/compute/ext/certificates/api/v1.1'

updated = '2012-01-19T00:00:00Z'

class CertificatesController
Bases: object

The x509 Certificates API controller for the OpenStack API.

create(*req*, *body=None*)
Create a certificate.

show(*req*, *id*)
Return certificate information.

3.7.30 The nova.api.openstack.compute.contrib.cloudpipe Module

Connect your vlan to the world.

class Cloudpipe(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds actions to create cloudpipe instances.

When running with the Vlan network mode, you need a mechanism to route from the public Internet to your vlans. This mechanism is known as a cloudpipe.

At the time of creating this class, only OpenVPN is supported. Support for a SSH Bastion host is forthcoming.

alias = 'os-cloudpipe'

get_resources()

name = 'Cloudpipe'

namespace = 'http://docs.openstack.org/compute/ext/cloudpipe/api/v1.1'

updated = '2011-12-16T00:00:00Z'

class CloudpipeController
Bases: object

Handle creating and listing cloudpipe instances.

create(*req*, *body*)
Create a new cloudpipe instance, if none exists.

Parameters: {cloudpipe: {'project_id': ''}}

index(*req*)
List running cloudpipe instances.

setup()
Ensure the keychains and folders exist.

3.7.31 The nova.api.openstack.compute.contrib.cloudpipe_update Module

class CloudpipeUpdateController
Bases: nova.api.openstack.wsgi.Controller

Handle updating the vpn ip/port for cloudpipe instances.

update(*req*, *id*, *body*)
Configure cloudpipe parameters for the project.

wsgi_actions = {'update': 'update'}

wsgi_extensions = []

class Cloudpipe_update (*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds the ability to set the vpn ip/port for cloudpipe instances.

alias = 'os-cloudpipe-update'

get_controller_extensions ()

name = 'CloudpipeUpdate'

namespace = 'http://docs.openstack.org/compute/ext/cloudpipe-update/api/v2'

updated = '2012-11-14T00:00:00Z'

3.7.32 The nova.api.openstack.compute.contrib.config_drive Module

Config Drive extension.

class Config_drive (*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Config Drive Extension.

alias = 'os-config-drive'

get_controller_extensions ()

name = 'ConfigDrive'

namespace = 'http://docs.openstack.org/compute/ext/config_drive/api/v1.1'

updated = '2012-07-16T00:00:00Z'

class Controller (*ext_mgr=None*, ***kwargs*)
Bases: nova.api.openstack.compute.servers.Controller

detail (*req*, *resp_obj*)

show (*req*, *resp_obj*, *id*)

wsgi_actions = {'createImage': '_action_create_image', 'revertResize': '_action_revert_resize', 'changePassword': '_

wsgi_extensions = [('detail', None), ('show', None)]

3.7.33 The nova.api.openstack.compute.contrib.console_auth_tokens Module

class ConsoleAuthTokensController (**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

show (*req*, *id*)
Checks a console auth token and returns the related connect info.

wsgi_actions = {}

wsgi_extensions = []

class Console_auth_tokens (*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Console token authentication support.

alias = 'os-console-auth-tokens'

get_resources()

name = 'ConsoleAuthTokens'

namespace = 'http://docs.openstack.org/compute/ext/consoles-auth-tokens/api/v2'

updated = '2013-08-13T00:00:00Z'

3.7.34 The nova.api.openstack.compute.contrib.console_output Module

class ConsoleOutputController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

get_console_output(*req*, *id*, *body*)
Get text console output.

wsgi_actions = {'os-getConsoleOutput': 'get_console_output'}

wsgi_extensions = []

class Console_output(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Console log output support, with tailing ability.

alias = 'os-console-output'

get_controller_extensions()

name = 'ConsoleOutput'

namespace = 'http://docs.openstack.org/compute/ext/os-console-output/api/v2'

updated = '2011-12-08T00:00:00Z'

3.7.35 The nova.api.openstack.compute.contrib.consoles Module

class Consoles(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Interactive Console support.

alias = 'os-consoles'

get_controller_extensions()

name = 'Consoles'

namespace = 'http://docs.openstack.org/compute/ext/os-consoles/api/v2'

updated = '2011-12-23T00:00:00Z'

class ConsolesController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

get_rdp_console(*req*, *id*, *body*)
Get text console output.

get_serial_console(*req*, *id*, *body*)
Get connection to a serial console.

get_spice_console(*req*, *id*, *body*)
Get spice connection information to access a server.

get_vnc_console(*req*, *id*, *body*)
Get vnc connection information to access a server.

wsgi_actions = {'os-getRDPConsole': 'get_rdp_console', 'os-getSPICEConsole': 'get_spice_console', 'os-getSerialCor

wsgi_extensions = []

3.7.36 The nova.api.openstack.compute.contrib.createserverext Module

class Createserverext(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended support to the Create Server v1.1 API.

alias = 'os-create-server-ext'

get_resources()

name = 'Createserverext'

namespace = 'http://docs.openstack.org/compute/ext/createserverext/api/v1.1'

updated = '2011-07-19T00:00:00Z'

3.7.37 The nova.api.openstack.compute.contrib.deferred_delete Module

The deferred instance delete extension.

class DeferredDeleteController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'restore': '_restore', 'forceDelete': '_force_delete'}

wsgi_extensions = []

class Deferred_delete(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Instance deferred delete.

alias = 'os-deferred-delete'

get_controller_extensions()

name = 'DeferredDelete'

namespace = 'http://docs.openstack.org/compute/ext/deferred-delete/api/v1.1'

updated = '2011-09-01T00:00:00Z'

3.7.38 The nova.api.openstack.compute.contrib.disk_config Module

Disk Config extension.

class Disk_config(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Disk Management Extension.

alias = 'OS-DCF'

get_controller_extensions()

name = 'DiskConfig'

namespace = 'http://docs.openstack.org/compute/ext/disk_config/api/v1.1'

updated = '2011-09-27T00:00:00Z'

class **ImageDiskConfigController**(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class **ServerDiskConfigController**(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

create(*req*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

update(*req*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', None), ('detail', None), ('update', None), ('_action_rebuild', 'rebuild'), (

disk_config_from_api(*value*)

disk_config_to_api(*value*)

3.7.39 The nova.api.openstack.compute.contrib.evacuate Module

class **Controller**(*ext_mgr*, **args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'evacuate': '_evacuate'}

wsgi_extensions = []

class **Evacuate**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Enables server evacuation.

alias = 'os-evacuate'

get_controller_extensions()

name = 'Evacuate'

namespace = 'http://docs.openstack.org/compute/ext/evacuate/api/v2'

updated = '2013-01-06T00:00:00Z'

3.7.40 The nova.api.openstack.compute.contrib.extended_availability_zone Module

The Extended Availability Zone Status API extension.

class ExtendedAZController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Extended_availability_zone(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended Availability Zone support.

alias = 'OS-EXT-AZ'

get_controller_extensions()

name = 'ExtendedAvailabilityZone'

namespace = 'http://docs.openstack.org/compute/ext/extended_availability_zone/api/v2'

updated = '2013-01-30T00:00:00Z'

3.7.41 The nova.api.openstack.compute.contrib.extended_evacuate_find_host Module

class Extended_evacuate_find_host(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Enables server evacuation without target host. Scheduler will select one to target.

alias = 'os-extended-evacuate-find-host'

name = 'ExtendedEvacuateFindHost'

namespace = 'http://docs.openstack.org/compute/ext/extended_evacuate_find_host/api/v2'

updated = '2014-02-12T00:00:00Z'

3.7.42 The nova.api.openstack.compute.contrib.extended_floating_ips Module

class Extended_floating_ips(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds optional fixed_address to the add floating IP command.

alias = 'os-extended-floating-ips'

name = 'ExtendedFloatingIps'

namespace = 'http://docs.openstack.org/compute/ext/extended_floating_ips/api/v2'

updated = '2013-04-19T00:00:00Z'

3.7.43 The nova.api.openstack.compute.contrib.extended_hypervisors Module

class Extended_hypervisors (*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended hypervisors support.

alias = 'os-extended-hypervisors'

name = 'ExtendedHypervisors'

namespace = 'http://docs.openstack.org/compute/ext/extended_hypervisors/api/v1.1'

updated = '2014-01-04T00:00:00Z'

3.7.44 The nova.api.openstack.compute.contrib.extended_ips Module

The Extended Ips API extension.

class ExtendedIpsController (**args*, ***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

detail (*req*, *resp_obj*)

show (*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Extended_ips (*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds type parameter to the ip list.

alias = 'OS-EXT-IPS'

get_controller_extensions ()

name = 'ExtendedIps'

namespace = 'http://docs.openstack.org/compute/ext/extended_ips/api/v1.1'

updated = '2013-01-06T00:00:00Z'

3.7.45 The nova.api.openstack.compute.contrib.extended_ips_mac Module

The Extended Ips API extension.

class ExtendedIpsMacController (**args*, ***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

detail (*req*, *resp_obj*)

show (*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class **Extended_ips_mac**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds mac address parameter to the ip list.

alias = 'OS-EXT-IPS-MAC'

get_controller_extensions()

name = 'ExtendedIpsMac'

namespace = 'http://docs.openstack.org/compute/ext/extended_ips_mac/api/v1.1'

updated = '2013-03-07T00:00:00Z'

3.7.46 The nova.api.openstack.compute.contrib.extended_networks Module

class **Extended_networks**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds additional fields to networks.

alias = 'os-extended-networks'

name = 'ExtendedNetworks'

namespace = 'http://docs.openstack.org/compute/ext/extended_networks/api/v2'

updated = '2014-05-09T00:00:00Z'

3.7.47 The nova.api.openstack.compute.contrib.extended_quotas Module

class **Extended_quotas**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds ability for admins to delete quota and optionally force the update Quota command.

alias = 'os-extended-quotas'

name = 'ExtendedQuotas'

namespace = 'http://docs.openstack.org/compute/ext/extended_quotas/api/v1.1'

updated = '2013-06-09T00:00:00Z'

3.7.48 The nova.api.openstack.compute.contrib.extended_rescue_with_image Module

class **Extended_rescue_with_image**(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Allow the user to specify the image to use for rescue.

alias = 'os-extended-rescue-with-image'

name = 'ExtendedRescueWithImage'

namespace = 'http://docs.openstack.org/compute/ext/extended_rescue_with_image/api/v2'

updated = '2014-01-04T00:00:00Z'

3.7.49 The nova.api.openstack.compute.contrib.extended_server_attributes Module

The Extended Server Attributes API extension.

class ExtendedServerAttributesController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Extended_server_attributes(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended Server Attributes support.

alias = 'OS-EXT-SRV-ATTR'

get_controller_extensions()

name = 'ExtendedServerAttributes'

namespace = 'http://docs.openstack.org/compute/ext/extended_status/api/v1.1'

updated = '2011-11-03T00:00:00Z'

3.7.50 The nova.api.openstack.compute.contrib.extended_services Module

class Extended_services(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended services support.

alias = 'os-extended-services'

name = 'ExtendedServices'

namespace = 'http://docs.openstack.org/compute/ext/extended_services/api/v2'

updated = '2013-05-17T00:00:00Z'

3.7.51 The nova.api.openstack.compute.contrib.extended_services_delete Module

class Extended_services_delete(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended services deletion support.

alias = 'os-extended-services-delete'

name = 'ExtendedServicesDelete'

namespace = 'http://docs.openstack.org/compute/ext/extended_services_delete/api/v2'

updated = '2013-12-10T00:00:00Z'

3.7.52 The nova.api.openstack.compute.contrib.extended_status Module

The Extended Status Admin API extension.

class ExtendedStatusController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Extended_status(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended Status support.

alias = 'OS-EXT-STS'

get_controller_extensions()

name = 'ExtendedStatus'

namespace = 'http://docs.openstack.org/compute/ext/extended_status/api/v1.1'

updated = '2011-11-03T00:00:00Z'

3.7.53 The nova.api.openstack.compute.contrib.extended_virtual_interfaces_net Module

class ExtendedServerVIFNetController
Bases: nova.api.openstack.wsgi.Controller

index(*req*, *resp_obj*, *server_id*)

wsgi_actions = {}

wsgi_extensions = [('index', None)]

class Extended_virtual_interfaces_net(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds network id parameter to the virtual interface list.

alias = 'OS-EXT-VIF-NET'

get_controller_extensions()

name = 'ExtendedVIFNet'

namespace = 'http://docs.openstack.org/compute/ext/extended-virtual-interfaces-net/api/v1.1'

updated = '2013-03-07T00:00:00Z'

3.7.54 The nova.api.openstack.compute.contrib.extended_volumes Module

The Extended Volumes API extension.

class ExtendedVolumesController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Extended_volumes(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Extended Volumes support.

alias = 'os-extended-volumes'

get_controller_extensions()

get_resources()

name = 'ExtendedVolumes'

namespace = 'http://docs.openstack.org/compute/ext/extended_volumes/api/v1.1'

updated = '2013-06-07T00:00:00Z'

3.7.55 The nova.api.openstack.compute.contrib.fixed_ips Module

class FixedIPController
Bases: object

action(*req*, *id*, *body*)

show(*req*, *id*)
Return data about the given fixed ip.

class Fixed_ips(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Fixed IPs support.

alias = 'os-fixed-ips'

get_resources()

name = 'FixedIPs'

namespace = 'http://docs.openstack.org/compute/ext/fixed_ips/api/v2'

updated = '2012-10-18T19:25:27Z'

3.7.56 The nova.api.openstack.compute.contrib.flavor_access Module

The flavor access extension.

class FlavorAccessController
Bases: object

The flavor access API controller for the OpenStack API.

index(*req*, *flavor_id*)

class `FlavorActionController`(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

The flavor access API controller for the OpenStack API.

`create`(*req*, *body*, *resp_obj*)

`detail`(*req*, *resp_obj*)

`show`(*req*, *resp_obj*, *id*)

wsgi_actions = {'removeTenantAccess': '_removeTenantAccess', 'addTenantAccess': '_addTenantAccess'}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

class `Flavor_access`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Flavor access support.

alias = 'os-flavor-access'

`get_controller_extensions`()

`get_resources`()

name = 'FlavorAccess'

namespace = 'http://docs.openstack.org/compute/ext/flavor_access/api/v2'

updated = '2012-08-01T00:00:00Z'

3.7.57 The nova.api.openstack.compute.contrib.flavor_disabled Module

The Flavor Disabled API extension.

class `FlavorDisabledController`(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

`create`(*req*, *resp_obj*, *body*)

`detail`(*req*, *resp_obj*)

`show`(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

class `Flavor_disabled`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Support to show the disabled status of a flavor.

alias = 'OS-FLV-DISABLED'

`get_controller_extensions`()

name = 'FlavorDisabled'

namespace = 'http://docs.openstack.org/compute/ext/flavor_disabled/api/v1.1'

updated = '2012-08-29T00:00:00Z'

3.7.58 The nova.api.openstack.compute.contrib.flavor_rxtx Module

The Flavor Rxtx API extension.

class **FlavorRxtxController**(*view_builder=None*)

Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

class **Flavor_rxtx**(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Support to show the rxtx status of a flavor.

alias = 'os-flavor-rxtx'

get_controller_extensions()

name = 'FlavorRxtx'

namespace = 'http://docs.openstack.org/compute/ext/flavor_rxtx/api/v1.1'

updated = '2012-08-29T00:00:00Z'

3.7.59 The nova.api.openstack.compute.contrib.flavor_swap Module

The Flavor Swap API extension.

class **FlavorSwapController**(*view_builder=None*)

Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

class **Flavor_swap**(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Support to show the swap status of a flavor.

alias = 'os-flavor-swap'

get_controller_extensions()

name = 'FlavorSwap'

namespace = 'http://docs.openstack.org/compute/ext/flavor_swap/api/v1.1'

updated = '2012-08-29T00:00:00Z'

3.7.60 The nova.api.openstack.compute.contrib.flavorextradata Module

The Flavor extra data extension

OpenStack API version 1.1 lists "name", "ram", "disk", "vcpus" as flavor attributes. This extension adds to that list:

- OS-FLV-EXT-DATA:ephemeral

class Flavorextradata(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Provide additional data for flavors.

alias = 'OS-FLV-EXT-DATA'

get_controller_extensions()

name = 'FlavorExtraData'

namespace = 'http://docs.openstack.org/compute/ext/flavor_extra_data/api/v1.1'

updated = '2011-09-14T00:00:00Z'

class FlavorextradataController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

3.7.61 The nova.api.openstack.compute.contrib.flavorextraspecs Module

The instance type extra specs extension.

class FlavorExtraSpecsController
Bases: object

The flavor extra specs API controller for the OpenStack API.

create(*req*, *flavor_id*, *body*)

delete(*req*, *flavor_id*, *id*)
Deletes an existing extra spec.

index(*req*, *flavor_id*)
Returns the list of extra specs for a given flavor.

show(*req*, *flavor_id*, *id*)
Return a single extra spec item.

update(*req*, *flavor_id*, *id*, *body*)

class Flavorextraspecs(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Instance type (flavor) extra specs.

alias = 'os-flavor-extra-specs'

get_resources()

name = 'FlavorExtraSpecs'

namespace = 'http://docs.openstack.org/compute/ext/flavor_extra_specs/api/v1.1'

updated = '2011-06-23T00:00:00Z'

3.7.62 The nova.api.openstack.compute.contrib.flavormanage Module

class FlavorManageController
Bases: nova.api.openstack.wsgi.Controller

The Flavor Lifecycle API controller for the OpenStack API.

wsgi_actions = {'create': '_create', 'delete': '_delete'}

wsgi_extensions = []

class Flavormanage(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Flavor create/delete API support.

alias = 'os-flavor-manage'

get_controller_extensions()

name = 'FlavorManage'

namespace = 'http://docs.openstack.org/compute/ext/flavor_manage/api/v1.1'

updated = '2012-01-19T00:00:00Z'

3.7.63 The nova.api.openstack.compute.contrib.floating_ip_dns Module

class FloatingIPDNSDomainController
Bases: object

DNS domain controller for OpenStack API.

delete(*req*, *id*)
Delete the domain identified by id.

index(*req*)
Return a list of available DNS domains.

update(*req*, *id*, *body*)
Add or modify domain entry.

class FloatingIPDNSEntryController
Bases: object

DNS Entry controller for OpenStack API.

delete(*req*, *domain_id*, *id*)
Delete the entry identified by req and id.

show(*req*, *domain_id*, *id*)
Return the DNS entry that corresponds to domain_id and id.

update(*req*, *domain_id*, *id*, *body*)
Add or modify dns entry.

class `Floating_ip_dns`(*ext_mgr*)
Bases: `nova.api.openstack.extensions.ExtensionDescriptor`

Floating IP DNS support.

`alias` = 'os-floating-ip-dns'

`get_resources`()

`name` = 'FloatingIpDns'

`namespace` = 'http://docs.openstack.org/ext/floating_ip_dns/api/v1.1'

`updated` = '2011-12-23T00:00:00Z'

3.7.64 The `nova.api.openstack.compute.contrib.floating_ip_pools` Module

class `FloatingIPPoolsController`
Bases: `object`

The Floating IP Pool API controller for the OpenStack API.

`index`(*req*)
Return a list of pools.

class `Floating_ip_pools`(*ext_mgr*)
Bases: `nova.api.openstack.extensions.ExtensionDescriptor`

Floating IPs support.

`alias` = 'os-floating-ip-pools'

`get_resources`()

`name` = 'FloatingIpPools'

`namespace` = 'http://docs.openstack.org/compute/ext/floating_ip_pools/api/v1.1'

`updated` = '2012-01-04T00:00:00Z'

3.7.65 The `nova.api.openstack.compute.contrib.floating_ips` Module

class `FloatingIPActionController`(*ext_mgr=None*, **args*, ***kwargs*)
Bases: `nova.api.openstack.wsgi.Controller`

`wsgi_actions` = {'removeFloatingIp': '_remove_floating_ip', 'addFloatingIp': '_add_floating_ip'}

`wsgi_extensions` = []

class `FloatingIPController`
Bases: `object`

The Floating IPs API controller for the OpenStack API.

`create`(*req*, *body=None*)

`delete`(*req*, *id*)

`index`(*req*)
Return a list of floating ips allocated to a project.

`show`(*req*, *id*)
Return data about the given floating ip.

class `Floating_ips`(*ext_mgr*)
Bases: `nova.api.openstack.extensions.ExtensionDescriptor`

Floating IPs support.

`alias` = 'os-floating-ips'

`get_controller_extensions`()

`get_resources`()

`name` = 'FloatingIps'

`namespace` = 'http://docs.openstack.org/compute/ext/floating_ips/api/v1.1'

`updated` = '2011-06-16T00:00:00Z'

`disassociate_floating_ip`(*self*, *context*, *instance*, *address*)

`get_instance_by_floating_ip_addr`(*self*, *context*, *address*)

3.7.66 The `nova.api.openstack.compute.contrib.floating_ips_bulk` Module

class `FloatingIPBulkController`
Bases: `object`

`create`(*req*, *body*)
Bulk create floating ips.

`index`(*req*)
Return a list of all floating ips.

`show`(*req*, *id*)
Return a list of all floating ips for a given host.

`update`(*req*, *id*, *body*)
Bulk delete floating IPs.

class `Floating_ips_bulk`(*ext_mgr*)
Bases: `nova.api.openstack.extensions.ExtensionDescriptor`

Bulk handling of Floating IPs.

`alias` = 'os-floating-ips-bulk'

`get_resources`()

`name` = 'FloatingIpsBulk'

`namespace` = 'http://docs.openstack.org/compute/ext/floating_ips_bulk/api/v2'

`updated` = '2012-10-29T19:25:27Z'

3.7.67 The `nova.api.openstack.compute.contrib.fping` Module

class `Fping`(*ext_mgr*)
Bases: `nova.api.openstack.extensions.ExtensionDescriptor`

Fping Management Extension.

`alias` = 'os-fping'

get_resources()

name = 'Fping'

namespace = 'http://docs.openstack.org/compute/ext/fping/api/v1.1'

updated = '2012-07-06T00:00:00Z'

class FpingController(*network_api=None*)
Bases: object

check_fping()

static fping(*ips*)

index(*req*)

show(*req*, *id*)

3.7.68 The nova.api.openstack.compute.contrib.hide_server_addresses Module

Extension for hiding server addresses in certain states.

class Controller(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Hide_server_addresses(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Support hiding server addresses in certain states.

alias = 'os-hide-server-addresses'

get_controller_extensions()

name = 'HideServerAddresses'

namespace = 'http://docs.openstack.org/compute/ext/hide_server_addresses/api/v1.1'

updated = '2012-12-11T00:00:00Z'

3.7.69 The nova.api.openstack.compute.contrib.hosts Module

The hosts admin extension.

class HostController
Bases: object

The Hosts API controller for the OpenStack API.

index(*req*)
Returns a dict in the format:

{'hosts': [{'host_name': 'some.host.name',

'service': 'cells',
'zone': 'internal'},
{'host_name': 'some.other.host.name',
'service': 'cells',
'zone': 'internal'},
{'host_name': 'some.celly.host.name',
'service': 'cells',
'zone': 'internal'},
{'host_name': 'console1.host.com',
'service': 'consoleauth',
'zone': 'internal'},
{'host_name': 'network1.host.com',
'service': 'network',
'zone': 'internal'},
{'host_name': 'netwwork2.host.com',
'service': 'network',
'zone': 'internal'},
{'host_name': 'compute1.host.com',
'service': 'compute',
'zone': 'nova'},
{'host_name': 'compute2.host.com',
'service': 'compute',
'zone': 'nova'},
{'host_name': 'sched1.host.com',
'service': 'scheduler',
'zone': 'internal'},
{'host_name': 'sched2.host.com',
'service': 'scheduler',
'zone': 'internal'},
{'host_name': 'vol1.host.com',
'service': 'volume',
'zone': 'internal'}]}

reboot(*req*, *id*)

show(*req*, *id*)

Shows the physical/usage resource given by hosts.

Parameters **id** – hostname

Returns

expected to use HostShowTemplate. ex.:

```
{'host': {'resource':D},..}
D: {'host': 'hostname','project': 'admin',
    'cpu': 1, 'memory_mb': 2048, 'disk_gb': 30}
```

shutdown(*req*, *id*)

startup(*req*, *id*)

update(*req*, *id*, *body*)
Updates a specified body.

Parameters **body** – example format {'status': 'enable', 'maintenance_mode': 'enable'}

class Hosts(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Admin-only host administration.

alias = 'os-hosts'

get_resources()

name = 'Hosts'

namespace = 'http://docs.openstack.org/compute/ext/hosts/api/v1.1'

updated = '2011-06-29T00:00:00Z'

3.7.70 The nova.api.openstack.compute.contrib.hypervisor_status Module

class Hypervisor_status(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Show hypervisor status.

alias = 'os-hypervisor-status'

name = 'HypervisorStatus'

namespace = 'http://docs.openstack.org/compute/ext/hypervisor_status/api/v1.1'

updated = '2014-04-17T00:00:00Z'

3.7.71 The nova.api.openstack.compute.contrib.hypervisors Module

The hypervisors admin extension.

class Hypervisors(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Admin-only hypervisor administration.

alias = 'os-hypervisors'

get_resources()

name = 'Hypervisors'

namespace = 'http://docs.openstack.org/compute/ext/hypervisors/api/v1.1'

updated = '2012-06-21T00:00:00Z'

class HypervisorsController(*ext_mgr*)
Bases: object

The Hypervisors API controller for the OpenStack API.

detail(*req*)

index(*req*)

search(*req*, *id*)

servers(*req*, *id*)

show(*req*, *id*)

statistics(*req*)

uptime(*req*, *id*)

3.7.72 The nova.api.openstack.compute.contrib.image_size Module

class ImageSizeController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Image_size(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds image size to image listings.

alias = 'OS-EXT-IMG-SIZE'

get_controller_extensions()

name = 'ImageSize'

namespace = 'http://docs.openstack.org/compute/ext/image_size/api/v1.1'

updated = '2013-02-19T00:00:00Z'

3.7.73 The nova.api.openstack.compute.contrib.instance_actions Module

class InstanceActionsController
Bases: nova.api.openstack.wsgi.Controller

index(*req*, *server_id*)
Returns the list of actions recorded for a given instance.

show(*req*, *server_id*, *id*)
Return data about the given instance action.

wsgi_actions = {}

wsgi_extensions = []

class Instance_actions(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

View a log of actions and events taken on an instance.

alias = 'os-instance-actions'

get_resources()

name = 'InstanceActions'

namespace = 'http://docs.openstack.org/compute/ext/instance-actions/api/v1.1'

updated = '2013-02-08T00:00:00Z'

3.7.74 The nova.api.openstack.compute.contrib.instance_usage_audit_log Module

class InstanceUsageAuditLogController
Bases: object

index(*req*)

show(*req*, *id*)

class Instance_usage_audit_log(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Admin-only Task Log Monitoring.

alias = 'os-instance_usage_audit_log'

get_resources()

name = 'OSInstanceUsageAuditLog'

namespace = 'http://docs.openstack.org/ext/services/api/v1.1'

updated = '2012-07-06T01:00:00Z'

3.7.75 The nova.api.openstack.compute.contrib.keypairs Module

Keypair management extension.

class Controller(*ext_mgr=None*, ***kwargs*)
Bases: nova.api.openstack.compute.servers.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {'createImage': '_action_create_image', 'revertResize': '_action_revert_resize', 'changePassword': '_

wsgi_extensions = [('show', None), ('detail', None)]

class KeypairController
Bases: object

Keypair API controller for the OpenStack API.

create(*req*, *body*)
Create or import keypair.

Sending name will generate a key and return private_key and fingerprint.

You can send a public_key to add an existing ssh key

params: keypair object with: name (required) - string public_key (optional) - string

delete(*req*, *id*)
Delete a keypair with a given name.

index(*req*)
List of keypairs for a user.

show(*req*, *id*)
Return data for the given key name.

class Keypairs(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Keypair Support.

alias = 'os-keypairs'

get_controller_extensions()

get_resources()

name = 'Keypairs'

namespace = 'http://docs.openstack.org/compute/ext/keypairs/api/v1.1'

updated = '2011-08-08T00:00:00Z'

3.7.76 The nova.api.openstack.compute.contrib.migrations Module

class Migrations(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Provide data on migrations.

alias = 'os-migrations'

get_resources()

name = 'Migrations'

namespace = 'http://docs.openstack.org/compute/ext/migrations/api/v2.0'

updated = '2013-05-30T00:00:00Z'

class MigrationsController
Bases: object

Controller for accessing migrations in OpenStack API.

index(*req*)
Return all migrations in progress.

authorize(*context*, *action_name*)

output(*migrations_obj*)
Returns the desired output of the API from an object.

From a MigrationsList's object this method returns a list of primitive objects with the only necessary fields.

3.7.77 The nova.api.openstack.compute.contrib.multinic Module

The multinic extension.

class Multinic(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Multiple network support.

alias = 'NMN'

get_controller_extensions()

name = 'Multinic'

namespace = 'http://docs.openstack.org/compute/ext/multinic/api/v1.1'

updated = '2011-06-09T00:00:00Z'

class MultinicController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'addFixedIp': '_add_fixed_ip', 'removeFixedIp': '_remove_fixed_ip'}

wsgi_extensions = []

3.7.78 The nova.api.openstack.compute.contrib.multiple_create Module

class Multiple_create(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Allow multiple create in the Create Server v1.1 API.

alias = 'os-multiple-create'

name = 'MultipleCreate'

namespace = 'http://docs.openstack.org/compute/ext/multiplecreate/api/v1.1'

updated = '2012-08-07T00:00:00Z'

3.7.79 The nova.api.openstack.compute.contrib.networks_associate Module

class NetworkAssociateActionController(*network_api=None*)
Bases: nova.api.openstack.wsgi.Controller

Network Association API Controller.

wsgi_actions = {'disassociate_project': '_disassociate_project_only', 'associate_host': '_associate_host', 'disassociate_

wsgi_extensions = []

class Networks_associate(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Network association support.

alias = 'os-networks-associate'

get_controller_extensions()

name = 'NetworkAssociationSupport'

namespace = 'http://docs.openstack.org/compute/ext/networks_associate/api/v2'

updated = '2012-11-19T00:00:00Z'

3.7.80 The nova.api.openstack.compute.contrib.os_networks Module

class NetworkController(*network_api=None*, *ext_mgr=None*)
Bases: nova.api.openstack.wsgi.Controller

add(*req*, *body*)

create(*req*, *body*)

delete(*req*, *id*)

index(*req*)

show(*req*, *id*)

wsgi_actions = {'disassociate': '_disassociate_host_and_project'}

wsgi_extensions = []

class Os_networks(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Admin-only Network Management Extension.

alias = 'os-networks'

get_resources()

name = 'Networks'

namespace = 'http://docs.openstack.org/compute/ext/os-networks/api/v1.1'

updated = '2011-12-23T00:00:00Z'

network_dict(*context*, *network*, *extended*)

3.7.81 The nova.api.openstack.compute.contrib.os_tenant_networks Module

class NetworkController(*network_api=None*)

Bases: object

create(*req*, *body*)

delete(*req*, *id*)

index(*req*)

show(*req*, *id*)

class Os_tenant_networks(*ext_mgr*)

Bases: nova.api.openstack.extensions.ExtensionDescriptor

Tenant-based Network Management Extension.

alias = 'os-tenant-networks'

get_resources()

name = 'OSTenantNetworks'

namespace = 'http://docs.openstack.org/compute/ext/os-tenant-networks/api/v2'

updated = '2012-03-07T14:46:43Z'

network_dict(*network*)

3.7.82 The nova.api.openstack.compute.contrib.preserve_ephemeral_rebuild Module

class Preserve_ephemeral_rebuild(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Allow preservation of the ephemeral partition on rebuild.

alias = 'os-preserve-ephemeral-rebuild'

name = 'PreserveEphemeralOnRebuild'

namespace = 'http://docs.openstack.org/compute/ext/preserve_ephemeral_rebuild/api/v2'

updated = '2013-12-17T00:00:00Z'

3.7.83 The nova.api.openstack.compute.contrib.quota_classes Module

class QuotaClassSetsController(*ext_mgr*)
Bases: nova.api.openstack.wsgi.Controller

show(*req*, *id*)

supported_quotas = []

update(*req*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

class Quota_classes(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Quota classes management support.

alias = 'os-quota-class-sets'

get_resources()

name = 'QuotaClasses'

namespace = 'http://docs.openstack.org/compute/ext/quota-classes-sets/api/v1.1'

updated = '2012-03-12T00:00:00Z'

3.7.84 The nova.api.openstack.compute.contrib.quotas Module

class QuotaSetsController(*ext_mgr*)
Bases: nova.api.openstack.wsgi.Controller

defaults(*req*, *id*)

delete(*req*, *id*)

show(*req*, *id*)

supported_quotas = []

update(*req*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

class `Quotas`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Quotas management support.

alias = 'os-quota-sets'

`get_resources`()

name = 'Quotas'

namespace = 'http://docs.openstack.org/compute/ext/quotas-sets/api/v1.1'

updated = '2011-08-08T00:00:00Z'

3.7.85 The nova.api.openstack.compute.contrib.rescue Module

The rescue mode extension.

class `Rescue`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Instance rescue mode.

alias = 'os-rescue'

`get_controller_extensions`()

name = 'Rescue'

namespace = 'http://docs.openstack.org/compute/ext/rescue/api/v1.1'

updated = '2011-08-18T00:00:00Z'

class `RescueController`(*ext_mgr*, **args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'unrescue': '_unrescue', 'rescue': '_rescue'}

wsgi_extensions = []

3.7.86 The nova.api.openstack.compute.contrib.scheduler_hints Module

class `SchedulerHintsController`(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

`create`(*req*, *body*)

wsgi_actions = {}

wsgi_extensions = [('create', None)]

class `Scheduler_hints`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Pass arbitrary key/value pairs to the scheduler.

alias = 'OS-SCH-HNT'

`get_controller_extensions`()

name = 'SchedulerHints'

namespace = 'http://docs.openstack.org/compute/ext/scheduler-hints/api/v2'

updated = '2011-07-19T00:00:00Z'

3.7.87 The nova.api.openstack.compute.contrib.security_group_default_rules Module

class SecurityGroupDefaultRulesController
Bases: nova.api.openstack.compute.contrib.security_groups.SecurityGroupControllerBase

create(*req*, *body*)

delete(*req*, *id*)

index(*req*)

show(*req*, *id*)

class Security_group_default_rules(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Default rules for security group support.

alias = 'os-security-group-default-rules'

get_resources()

name = 'SecurityGroupDefaultRules'

namespace = 'http://docs.openstack.org/compute/ext/securitygroupdefaultrules/api/v1.1'

updated = '2013-02-05T00:00:00Z'

3.7.88 The nova.api.openstack.compute.contrib.security_groups Module

The security groups extension.

class SecurityGroupActionController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'removeSecurityGroup': '_removeSecurityGroup', 'addSecurityGroup': '_addSecurityGroup'}

wsgi_extensions = []

class SecurityGroupController
Bases: nova.api.openstack.compute.contrib.security_groups.SecurityGroupControllerBase

The Security group API controller for the OpenStack API.

create(*req*, *body*)
Creates a new security group.

delete(*req*, *id*)
Delete a security group.

index(*req*)
Returns a list of security groups.

show(*req*, *id*)
Return data about the given security group.

update(*req*, *id*, *body*)
Update a security group.

class SecurityGroupControllerBase
Bases: object

Base class for Security Group controllers.

class SecurityGroupRulesController
Bases: nova.api.openstack.compute.contrib.security_groups.SecurityGroupControllerBase

create(*req*, *body*)

delete(*req*, *id*)

class SecurityGroupsOutputController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', None), ('detail', None)]

class Security_groups(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Security group support.

alias = 'os-security-groups'

get_controller_extensions()

get_resources()

name = 'SecurityGroups'

namespace = 'http://docs.openstack.org/compute/ext/securitygroups/api/v1.1'

updated = '2013-05-28T00:00:00Z'

class ServerSecurityGroupController
Bases: nova.api.openstack.compute.contrib.security_groups.SecurityGroupControllerBase

index(*req*, *server_id*)
Returns a list of security groups for the given instance.

translate_exceptions(**args*, ***kwds*)
Translate nova exceptions to http exceptions.

3.7.89 The nova.api.openstack.compute.contrib.server_diagnostics Module

class ServerDiagnosticsController
Bases: object

index(*req*, *server_id*)

class Server_diagnostics(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Allow Admins to view server diagnostics through server action.

alias = 'os-server-diagnostics'

get_resources()

name = 'ServerDiagnostics'

namespace = 'http://docs.openstack.org/compute/ext/server-diagnostics/api/v1.1'

updated = '2011-12-21T00:00:00Z'

3.7.90 The nova.api.openstack.compute.contrib.server_external_events Module

class ServerExternalEventsController
Bases: nova.api.openstack.wsgi.Controller

create(*req*, *body*)
Creates a new instance event.

wsgi_actions = {}

wsgi_extensions = []

class Server_external_events(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Server External Event Triggers.

alias = 'os-server-external-events'

get_resources()

name = 'ServerExternalEvents'

namespace = 'http://docs.openstack.org/compute/ext/server-external-events/api/v2'

updated = '2014-02-18T00:00:00Z'

3.7.91 The nova.api.openstack.compute.contrib.server_group_quotas Module

class ExtendedLimitsController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

index(*req*, *resp_obj*)

wsgi_actions = {}

wsgi_extensions = [('index', None)]

class Server_group_quotas(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds quota support to server groups.

alias = 'os-server-group-quotas'

get_controller_extensions()

name = 'ServerGroupQuotas'

namespace = 'http://docs.openstack.org/compute/ext/server-group-quotas/api/v2'

updated = '2014-07-25T00:00:00Z'

3.7.92 The nova.api.openstack.compute.contrib.server_groups Module

The Server Group API Extension.

class ServerGroupController(*ext_mgr*)
Bases: nova.api.openstack.wsgi.Controller

The Server group API controller for the OpenStack API.

create(*req*, *body*)
Creates a new server group.

delete(*req*, *id*)
Delete an server group.

index(*req*)
Returns a list of server groups.

show(*req*, *id*)
Return data about the given server group.

wsgi_actions = {}

wsgi_extensions = []

class Server_groups(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Server group support.

alias = 'os-server-groups'

get_resources()

name = 'ServerGroups'

namespace = 'http://docs.openstack.org/compute/ext/servergroups/api/v2'

updated = '2013-06-20T00:00:00Z'

3.7.93 The nova.api.openstack.compute.contrib.server_list_multi_status Module

class Server_list_multi_status(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Allow to specify multiple status values concurrently in the servers list API..

alias = 'os-server-list-multi-status'

name = 'ServerListMultiStatus'

namespace = 'http://docs.openstack.org/compute/ext/os-server-list-multi-status/api/v2'

updated = '2014-05-11T00:00:00Z'

3.7.94 The nova.api.openstack.compute.contrib.server_password Module

The server password extension.

class ServerPasswordController
Bases: object

The Server Password API controller for the OpenStack API.

delete(*req*, *server_id*)

index(*req*, *server_id*)

class Server_password(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Server password support.

alias = 'os-server-password'

get_resources()

name = 'ServerPassword'

namespace = 'http://docs.openstack.org/compute/ext/server-password/api/v2'

updated = '2012-11-29T00:00:00Z'

3.7.95 The nova.api.openstack.compute.contrib.server_sort_keys Module

class Server_sort_keys(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Add sort keys and directions to the Server GET v2 API.

alias = 'os-server-sort-keys'

name = 'ServerSortKeys'

namespace = 'http://docs.openstack.org/compute/ext/server_sort_keys/api/v2'

updated = '2014-05-22T00:00:00Z'

3.7.96 The nova.api.openstack.compute.contrib.server_start_stop Module

class ServerStartStopActionController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'os-stop': '_stop_server', 'os-start': '_start_server'}

wsgi_extensions = []

class Server_start_stop(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Start/Stop instance compute API support.

alias = 'os-server-start-stop'

get_controller_extensions()

name = 'ServerStartStop'

namespace = 'http://docs.openstack.org/compute/ext/servers/api/v1.1'

updated = '2012-01-23T00:00:00Z'

3.7.97 The nova.api.openstack.compute.contrib.server_usage Module

class ServerUsageController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class Server_usage(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Adds launched_at and terminated_at on Servers.

alias = 'OS-SRV-USG'

get_controller_extensions()

name = 'ServerUsage'

namespace = 'http://docs.openstack.org/compute/ext/server_usage/api/v1.1'

updated = '2013-04-29T00:00:00Z'

3.7.98 The nova.api.openstack.compute.contrib.services Module

class ServiceController(*ext_mgr=None*, **args*, ***kwargs*)
Bases: object

delete(*req*, *id*)
Deletes the specified service.

index(*req*)
Return a list of all running services.

update(*req*, *id*, *body*)
Enable/Disable scheduling for a service.

class Services(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Services support.

alias = 'os-services'

get_resources()

name = 'Services'

namespace = 'http://docs.openstack.org/compute/ext/services/api/v2'

updated = '2012-10-28T00:00:00Z'

3.7.99 The nova.api.openstack.compute.contrib.shelve Module

The shelved mode extension.

class Shelve(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Instance shelve mode.

alias = 'os-shelve'

get_controller_extensions()

name = 'Shelve'

namespace = 'http://docs.openstack.org/compute/ext/shelve/api/v1.1'

updated = '2013-04-06T00:00:00Z'

class ShelveController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'unshelve': '_unshelve', 'shelveOffload': '_shelve_offload', 'shelve': '_shelve'}

wsgi_extensions = []

3.7.100 The nova.api.openstack.compute.contrib.simple_tenant_usage Module

class SimpleTenantUsageController
Bases: object

index(*req*)
Retrieve tenant_usage for all tenants.

show(*req*, *id*)
Retrieve tenant_usage for a specified tenant.

class Simple_tenant_usage(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Simple tenant usage extension.

alias = 'os-simple-tenant-usage'

get_resources()

name = 'SimpleTenantUsage'

namespace = 'http://docs.openstack.org/compute/ext/os-simple-tenant-usage/api/v1.1'

updated = '2011-08-19T00:00:00Z'

parse_strtime(*dstr*, *fmt*)

3.7.101 The nova.api.openstack.compute.contrib.used_limits Module

class UsedLimitsController(*ext_mgr*)
Bases: nova.api.openstack.wsgi.Controller

index(*req*, *resp_obj*)

wsgi_actions = {}

wsgi_extensions = [('index', None)]

class `Used_limits`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Provide data on limited resources that are being used.

alias = 'os-used-limits'

`get_controller_extensions`()

name = 'UsedLimits'

namespace = 'http://docs.openstack.org/compute/ext/used_limits/api/v1.1'

updated = '2012-07-13T00:00:00Z'

3.7.102 The nova.api.openstack.compute.contrib.used_limits_for_admin Module

class `Used_limits_for_admin`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Provide data to admin on limited resources used by other tenants.

alias = 'os-used-limits-for-admin'

name = 'UsedLimitsForAdmin'

namespace = 'http://docs.openstack.org/compute/ext/used_limits_for_admin/api/v1.1'

updated = '2013-05-02T00:00:00Z'

3.7.103 The nova.api.openstack.compute.contrib.user_data Module

class `User_data`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Add user_data to the Create Server v1.1 API.

alias = 'os-user-data'

name = 'UserData'

namespace = 'http://docs.openstack.org/compute/ext/userdata/api/v1.1'

updated = '2012-08-07T00:00:00Z'

3.7.104 The nova.api.openstack.compute.contrib.user_quotas Module

class `User_quotas`(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Project user quota support.

alias = 'os-user-quotas'

name = 'UserQuotas'

namespace = 'http://docs.openstack.org/compute/ext/user_quotas/api/v1.1'

updated = '2013-07-18T00:00:00Z'

3.7.105 The nova.api.openstack.compute.contrib.virtual_interfaces Module

The virtual interfaces extension.

class ServerVirtualInterfaceController
Bases: object

The instance VIF API controller for the OpenStack API.

index(*req*, *server_id*)
Returns the list of VIFs for a given instance.

class Virtual_interfaces(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Virtual interface support.

alias = 'os-virtual-interfaces'

get_resources()

name = 'VirtualInterfaces'

namespace = 'http://docs.openstack.org/compute/ext/virtual_interfaces/api/v1.1'

updated = '2011-08-17T00:00:00Z'

3.7.106 The nova.api.openstack.compute.contrib.volume_attachment_update Module

class Volume_attachment_update(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Support for updating a volume attachment.

alias = 'os-volume-attachment-update'

name = 'VolumeAttachmentUpdate'

namespace = 'http://docs.openstack.org/compute/ext/os-volume-attachment-update/api/v2'

updated = '2013-06-20T00:00:00Z'

3.7.107 The nova.api.openstack.compute.contrib.volumes Module

The volumes extension.

class SnapshotController
Bases: nova.api.openstack.wsgi.Controller

The Snapshots API controller for the OpenStack API.

create(*req*, *body*)
Creates a new snapshot.

delete(*req*, *id*)
Delete a snapshot.

detail(*req*)
Returns a detailed list of snapshots.

index(*req*)
Returns a summary list of snapshots.

show(*req*, *id*)
Return data about the given snapshot.

wsgi_actions = {}

wsgi_extensions = []

class VolumeAttachmentController(*ext_mgr=None*)
Bases: nova.api.openstack.wsgi.Controller

The volume attachment API controller for the OpenStack API.

A child resource of the server. Note that we use the volume id as the ID of the attachment (though this is not guaranteed externally)

create(*req*, *server_id*, *body*)
Attach a volume to an instance.

delete(*req*, *server_id*, *id*)
Detach a volume from an instance.

index(*req*, *server_id*)
Returns the list of volume attachments for a given instance.

show(*req*, *server_id*, *id*)
Return data about the given volume attachment.

update(*req*, *server_id*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

class VolumeController
Bases: nova.api.openstack.wsgi.Controller

The Volumes API controller for the OpenStack API.

create(*req*, *body*)
Creates a new volume.

delete(*req*, *id*)
Delete a volume.

detail(*req*)
Returns a detailed list of volumes.

index(*req*)
Returns a summary list of volumes.

show(*req*, *id*)
Return data about the given volume.

wsgi_actions = {}

wsgi_extensions = []

class Volumes(*ext_mgr*)
Bases: nova.api.openstack.extensions.ExtensionDescriptor

Volumes support.

alias = 'os-volumes'

get_resources()

name = 'Volumes'

namespace = 'http://docs.openstack.org/compute/ext/volumes/api/v1.1'

updated = '2011-03-25T00:00:00Z'

3.7.108 The nova.api.openstack.compute.extensions Module

class ExtensionManager
Bases: nova.api.openstack.extensions.ExtensionManager

3.7.109 The nova.api.openstack.compute.flavors Module

class Controller(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

Flavor controller for the OpenStack API.

detail(*req*)
Return all flavors in detail.

index(*req*)
Return all flavors in brief.

show(*req*, *id*)
Return data about the given flavor id.

wsgi_actions = {}

wsgi_extensions = []

create_resource()

3.7.110 The nova.api.openstack.compute.image_metadata Module

class Controller
Bases: object

The image metadata API controller for the OpenStack API.

create(*req*, *image_id*, *body*)

delete(*req*, *image_id*, *id*)

index(*req*, *image_id*)
Returns the list of metadata for a given instance.

show(*req*, *image_id*, *id*)

update(*req*, *image_id*, *id*, *body*)

update_all(*req*, *image_id*, *body*)

create_resource()

3.7.111 The nova.api.openstack.compute.images Module

class Controller(**kwargs)
Bases: nova.api.openstack.wsgi.Controller

Base controller for retrieving/displaying images.

create(*args, **kwargs)

delete(req, id)
Delete an image, if allowed.

Parameters

- **req** – *wsgi.Request* object
- **id** – Image identifier (integer)

detail(req)
Return a detailed index listing of images available to the request.

Parameters **req** – *wsgi.Request* object.

index(req)
Return an index listing of images available to the request.

Parameters **req** – *wsgi.Request* object

show(req, id)
Return detailed information about a specific image.

Parameters

- **req** – *wsgi.Request* object
- **id** – Image identifier

wsgi_actions = {}

wsgi_extensions = []

create_resource()

3.7.112 The nova.api.openstack.compute.ips Module

class Controller(**kwargs)
Bases: nova.api.openstack.wsgi.Controller

The servers addresses API controller for the OpenStack API.

index(req, server_id)

show(req, server_id, id)

wsgi_actions = {}

wsgi_extensions = []

create_resource()

3.7.113 The nova.api.openstack.compute.limits Module

Module dedicated functions/classes dealing with rate limiting requests.

This module handles rate liming at a per-user level, so it should not be used to prevent intentional Denial of Service attacks, as we can assume a DOS can easily come through multiple user accounts. DOS protection should be done at a different layer. Instead this module should be used to protect against unintentional user actions. With that in mind the limits set here should be high enough as to not rate-limit any intentional actions.

To find good rate-limit values, check how long requests are taking (see logs) in your environment to assess your capabilities and multiply out to get figures.

NOTE: As the rate-limiting here is done in memory, this only works per process (each process will have its own rate limiting counter).

class Limit (*verb*, *uri*, *regex*, *value*, *unit*)
Bases: object

Stores information about a limit for HTTP requests.

UNITS = {3600: 'HOUR', 1: 'SECOND', 86400: 'DAY', 60: 'MINUTE'}

display()
Return a useful representation of this class.

display_unit()
Display the string name of the unit.

class Limiter (*limits*, ***kwargs*)
Bases: object

Rate-limit checking class which handles limits in memory.

check_for_delay (*verb*, *url*, *username=None*)
Check the given verb/user/user triplet for limit.

@return: Tuple of delay (in seconds) and error message (or None, None)

get_limits (*username=None*)
Return the limits for a given user.

static parse_limits (*limits*)
Convert a string into a list of Limit instances. This implementation expects a semicolon-separated sequence of parenthesized groups, where each group contains a comma-separated sequence consisting of HTTP method, user-readable URI, a URI reg-exp, an integer number of requests which can be made, and a unit of measure. Valid values for the latter are "SECOND", "MINUTE", "HOUR", and "DAY".

@return: List of Limit instances.

class LimitsController
Bases: object

Controller for accessing limits in the OpenStack API.

create (*req*, *body*)
Create a new limit.

delete (*req*, *id*)
Delete the limit.

index (*req*)
Return all global and rate limit information.

show(*req*, *id*)
Show limit information.

update(*req*, *id*, *body*)
Update existing limit.

class RateLimitingMiddleware(*application*, *limits=None*, *limiter=None*, ***kwargs*)
Bases: nova.wsgi.Middleware

Rate-limits requests passing through this middleware. All limit information is stored in memory for this implementation.

class WsgiLimiter(*limits=None*)
Bases: object

Rate-limit checking from a WSGI application. Uses an in-memory *Limiter*.

To use, POST /<username> with JSON data such as:

```
{
    "verb" : GET,
    "path" : "/servers"
}
```

and receive a 204 No Content, or a 403 Forbidden with an X-Wait-Seconds header containing the number of seconds to wait before the action would succeed.

class WsgiLimiterProxy(*limiter_address*)
Bases: object

Rate-limit requests based on answers from a remote source.

check_for_delay(*verb*, *path*, *username=None*)

static parse_limits(*limits*)
Ignore a limits string–simply doesn't apply for the limit proxy.

@return: Empty list.

create_resource()

3.7.114 The nova.api.openstack.compute.plugins.v3.access_ips Module

class AccessIPs(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Access IPs support.

alias = 'os-access-ips'

get_controller_extensions()

get_resources()

get_server_create_schema()

get_server_rebuild_schema()

get_server_update_schema()

name = 'AccessIPs'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

server_rebuild(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

server_update(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

v4_key = 'accessIPv4'

v6_key = 'accessIPv6'

version = 1

class **AccessIPsController**(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

rebuild(*req*, *resp_obj*, *id*, *body*)

show(*req*, *resp_obj*, *id*)

update(*req*, *resp_obj*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('rebuild', 'rebuild'), ('detail', None), ('update', None)]

3.7.115 The nova.api.openstack.compute.plugins.v3.admin_actions Module

class **AdminActions**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enable admin-only server actions

Actions include: resetNetwork, injectNetworkInfo, os-resetState

alias = 'os-admin-actions'

get_controller_extensions()

get_resources()

name = 'AdminActions'

version = 1

class **AdminActionsController**(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'resetNetwork': '_reset_network', 'os-resetState': '_reset_state', 'injectNetworkInfo': '_inject_netwo

wsgi_extensions = []

3.7.116 The nova.api.openstack.compute.plugins.v3.admin_password Module

class **AdminPassword**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin password management support.

alias = 'os-admin-password'

get_controller_extensions()

get_resources()

name = 'AdminPassword'

version = 1

class AdminPasswordController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

change_password(**args*, ***kwargs*)

wsgi_actions = {'changePassword': 'change_password'}

wsgi_extensions = []

3.7.117 The nova.api.openstack.compute.plugins.v3.agents Module

class AgentController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

The agent is talking about guest agent.The host can use this for things like accessing files on the disk, configuring networking, or running other applications/scripts in the guest while it is running. Typically this uses some hypervisor-specific transport to avoid being dependent on a working network configuration. Xen, VMware, and VirtualBox have guest agents,although the Xen driver is the only one with an implementation for managing them in openstack. KVM doesn't really have a concept of a guest agent (although one could be written).

You can find the design of agent update in this link: http://wiki.openstack.org/AgentUpdate and find the code in nova.virt.xenapi.vmops.VMOps._boot_new_instance. In this design We need update agent in guest from host, so we need some interfaces to update the agent info in host.

You can find more information about the design of the GuestAgent in the following link: http://wiki.openstack.org/GuestAgent http://wiki.openstack.org/GuestAgentXenStoreCommunication

create(**args*, ***kwargs*)
Creates a new agent build.

delete(**args*, ***kwargs*)
Deletes an existing agent build.

index(**args*, ***kwargs*)
Return a list of all agent builds. Filter by hypervisor.

update(**args*, ***kwargs*)
Update an existing agent build.

wsgi_actions = {}

wsgi_extensions = []

class Agents(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Agents support.

alias = 'os-agents'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'Agents'

version = 1

3.7.118 The nova.api.openstack.compute.plugins.v3.aggregates Module

The Aggregate admin API extension.

class AggregateController
Bases: nova.api.openstack.wsgi.Controller

The Host Aggregates API controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates an aggregate, given its name and optional availability zone.

delete(**args*, ***kwargs*)
Removes an aggregate by id.

index(**args*, ***kwargs*)
Returns a list a host aggregate's id, name, availability_zone.

show(**args*, ***kwargs*)
Shows the details of an aggregate, hosts and metadata included.

update(**args*, ***kwargs*)
Updates the name and/or availability_zone of given aggregate.

wsgi_actions = {'add_host': '_add_host', 'set_metadata': '_set_metadata', 'remove_host': '_remove_host'}

wsgi_extensions = []

class Aggregates(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin-only aggregate administration.

alias = 'os-aggregates'

get_controller_extensions()

get_resources()

name = 'Aggregates'

version = 1

3.7.119 The nova.api.openstack.compute.plugins.v3.assisted_volume_snapshots Module

The Assisted volume snapshots extension.

class AssistedVolumeSnapshots(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Assisted volume snapshots.

alias = 'os-assisted-volume-snapshots'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'AssistedVolumeSnapshots'

version = 1

class AssistedVolumeSnapshotsController
Bases: nova.api.openstack.wsgi.Controller

The Assisted volume snapshots API controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates a new snapshot.

delete(**args*, ***kwargs*)
Delete a snapshot.

wsgi_actions = {}

wsgi_extensions = []

3.7.120 The nova.api.openstack.compute.plugins.v3.attach_interfaces Module

The instance interfaces extension.

class AttachInterfaces(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Attach interface support.

alias = 'os-attach-interfaces'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'AttachInterfaces'

version = 1

class InterfaceAttachmentController
Bases: nova.api.openstack.wsgi.Controller

The interface attachment API controller for the OpenStack API.

create(**args*, ***kwargs*)
Attach an interface to an instance.

delete(**args*, ***kwargs*)
Detach an interface from an instance.

index(**args*, ***kwargs*)
Returns the list of interface attachments for a given instance.

show(**args*, ***kwargs*)
Return data about the given interface attachment.

wsgi_actions = {}

wsgi_extensions = []

3.7.121 The nova.api.openstack.compute.plugins.v3.availability_zone Module

class AvailabilityZone(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

1.Add availability_zone to the Create Server API.

2.Add availability zones describing.

alias = 'os-availability-zone'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

get_server_create_schema()

name = 'AvailabilityZone'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

class AvailabilityZoneController
Bases: nova.api.openstack.wsgi.Controller

The Availability Zone API controller for the OpenStack API.

detail(**args*, ***kwargs*)
Returns a detailed list of availability zone.

index(**args*, ***kwargs*)
Returns a summary list of availability zone.

wsgi_actions = {}

wsgi_extensions = []

3.7.122 The nova.api.openstack.compute.plugins.v3.baremetal_nodes Module

The bare-metal admin extension.

class BareMetalNodeController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

The Bare-Metal Node API controller for the OpenStack API.

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {'add_interface': '_add_interface', 'remove_interface': '_remove_interface'}

wsgi_extensions = []

class BareMetalNodes(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin-only bare-metal node administration.

alias = 'os-baremetal-nodes'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'BareMetalNodes'

version = 1

3.7.123 The nova.api.openstack.compute.plugins.v3.block_device_mapping Module

The block device mappings extension.

class BlockDeviceMapping(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Block device mapping boot support.

alias = 'os-block-device-mapping'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'BlockDeviceMapping'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

3.7.124 The nova.api.openstack.compute.plugins.v3.block_device_mapping_v1 Module

The legacy block device mappings extension.

class BlockDeviceMappingV1(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Block device mapping boot support.

alias = 'os-block-device-mapping-v1'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'BlockDeviceMappingV1'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

3.7.125 The nova.api.openstack.compute.plugins.v3.cells Module

The cells extension.

class Cells(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enables cells-related functionality such as adding neighbor cells, listing neighbor cells, and getting the capabilities of the local cell.

alias = 'os-cells'

get_controller_extensions()

get_resources()

name = 'Cells'

version = 1

class CellsController
Bases: nova.api.openstack.wsgi.Controller

Controller for Cell resources.

capacities(**args*, ***kwargs*)
Return capacities for a given cell or all cells.

create(**args*, ***kwargs*)
Create a child cell entry.

delete(**args*, ***kwargs*)
Delete a child or parent cell entry. 'id' is a cell name.

detail(**args*, ***kwargs*)
Return all cells in detail.

index(**args*, ***kwargs*)
Return all cells in brief.

info(**args*, ***kwargs*)
Return name and capabilities for this cell.

show(**args*, ***kwargs*)
Return data about the given cell name. 'id' is a cell name.

sync_instances(**args*, ***kwargs*)
Tell all cells to sync instance info.

update(**args*, ***kwargs*)
Update a child cell entry. 'id' is the cell name to update.

wsgi_actions = {}

wsgi_extensions = []

3.7.126 The nova.api.openstack.compute.plugins.v3.certificates Module

class Certificates(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Certificates support.

alias = 'os-certificates'

get_controller_extensions()

get_resources()

name = 'Certificates'

version = 1

class CertificatesController
Bases: nova.api.openstack.wsgi.Controller

The x509 Certificates API controller for the OpenStack API.

create(**args*, ***kwargs*)
Create a certificate.

show(**args*, ***kwargs*)
Return certificate information.

wsgi_actions = {}

wsgi_extensions = []

3.7.127 The nova.api.openstack.compute.plugins.v3.cloudpipe Module

Connect your vlan to the world.

class Cloudpipe(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Adds actions to create cloudpipe instances.

When running with the Vlan network mode, you need a mechanism to route from the public Internet to your vlans. This mechanism is known as a cloudpipe.

At the time of creating this class, only OpenVPN is supported. Support for a SSH Bastion host is forthcoming.

alias = 'os-cloudpipe'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'Cloudpipe'

version = 1

class CloudpipeController
Bases: nova.api.openstack.wsgi.Controller

Handle creating and listing cloudpipe instances.

create(**args*, ***kwargs*)
Create a new cloudpipe instance, if none exists.

Parameters: {cloudpipe: {'project_id': ''}}

index(**args*, ***kwargs*)
List running cloudpipe instances.

setup()
Ensure the keychains and folders exist.

update(**args*, ***kwargs*)
Configure cloudpipe parameters for the project.

wsgi_actions = {}

wsgi_extensions = []

3.7.128 The nova.api.openstack.compute.plugins.v3.config_drive Module

Config Drive extension.

class ConfigDrive(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Config Drive Extension.

alias = 'os-config-drive'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'ConfigDrive'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

class ConfigDriveController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

3.7.129 The nova.api.openstack.compute.plugins.v3.console_auth_tokens Module

class ConsoleAuthTokens(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Console token authentication support.

alias = 'os-console-auth-tokens'

get_controller_extensions()

get_resources()

name = 'ConsoleAuthTokens'

version = 1

class ConsoleAuthTokensController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

show(**args*, ***kwargs*)
Checks a console auth token and returns the related connect info.

wsgi_actions = {}

wsgi_extensions = []

3.7.130 The nova.api.openstack.compute.plugins.v3.console_output Module

class ConsoleOutput(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Console log output support, with tailing ability.

alias = 'os-console-output'

get_controller_extensions()

get_resources()

name = 'ConsoleOutput'

version = 1

class ConsoleOutputController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

get_console_output(**args*, ***kwargs*)
Get text console output.

wsgi_actions = {'os-getConsoleOutput': 'get_console_output'}

wsgi_extensions = []

3.7.131 The nova.api.openstack.compute.plugins.v3.consoles Module

class Consoles(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Consoles.

alias = 'os-consoles'

get_controller_extensions()

get_resources()

name = 'Consoles'

version = 1

class ConsolesController
Bases: nova.api.openstack.wsgi.Controller

The Consoles controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates a new console.

delete(**args*, ***kwargs*)
Deletes a console.

index(**args*, ***kwargs*)
Returns a list of consoles for this instance.

show(**args*, ***kwargs*)
Shows in-depth information on a specific console.

wsgi_actions = {}

wsgi_extensions = []

3.7.132 The nova.api.openstack.compute.plugins.v3.create_backup Module

class CreateBackup(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Create a backup of a server.

alias = 'os-create-backup'

get_controller_extensions()

get_resources()

name = 'CreateBackup'

version = 1

class CreateBackupController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'createBackup': '_create_backup'}

wsgi_extensions = []

3.7.133 The nova.api.openstack.compute.plugins.v3.deferred_delete Module

The deferred instance delete extension.

class DeferredDelete(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Instance deferred delete.

alias = 'os-deferred-delete'

get_controller_extensions()

get_resources()

name = 'DeferredDelete'

version = 1

class DeferredDeleteController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'restore': '_restore', 'forceDelete': '_force_delete'}

wsgi_extensions = []

3.7.134 The nova.api.openstack.compute.plugins.v3.disk_config Module

Disk Config extension.

class DiskConfig(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Disk Management Extension.

alias = 'os-disk-config'

get_controller_extensions()

get_resources()

get_server_create_schema()

get_server_rebuild_schema()

get_server_resize_schema()

get_server_update_schema()

name = 'DiskConfig'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

server_rebuild(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

server_resize(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

server_update(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

version = 1

class ImageDiskConfigController(*view_builder=None*)

Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class ServerDiskConfigController(*view_builder=None*)

Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

update(*req*, *resp_obj*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', None), ('detail', None), ('update', None), ('_action_rebuild', 'rebuild')]

disk_config_from_api(*value*)

disk_config_to_api(*value*)

3.7.135 The nova.api.openstack.compute.plugins.v3.evacuate Module

class Evacuate(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enables server evacuation.

alias = 'os-evacuate'

get_controller_extensions()

get_resources()

name = 'Evacuate'

version = 1

class EvacuateController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'evacuate': '_evacuate'}

wsgi_extensions = []

3.7.136 The nova.api.openstack.compute.plugins.v3.extended_availability_zone Module

The Extended Availability Zone Status API extension.

class ExtendedAZController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class ExtendedAvailabilityZone(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Extended Availability Zone support.

alias = 'os-extended-availability-zone'

get_controller_extensions()

get_resources()

name = 'ExtendedAvailabilityZone'

version = 1

3.7.137 The nova.api.openstack.compute.plugins.v3.extended_server_attributes Module

The Extended Server Attributes API extension.

class ExtendedServerAttributes(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Extended Server Attributes support.

alias = 'os-extended-server-attributes'

get_controller_extensions()

get_resources()

name = 'ExtendedServerAttributes'

version = 1

class ExtendedServerAttributesController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show (*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

3.7.138 The nova.api.openstack.compute.plugins.v3.extended_status Module

The Extended Status Admin API extension.

class ExtendedStatus (*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Extended Status support.

alias = 'os-extended-status'

get_controller_extensions ()

get_resources ()

name = 'ExtendedStatus'

version = 1

class ExtendedStatusController (**args*, ***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

detail (*req*, *resp_obj*)

show (*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

3.7.139 The nova.api.openstack.compute.plugins.v3.extended_volumes Module

The Extended Volumes API extension.

class ExtendedVolumes (*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Extended Volumes support.

alias = 'os-extended-volumes'

get_controller_extensions ()

get_resources ()

name = 'ExtendedVolumes'

version = 1

class ExtendedVolumesController (**args*, ***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

detail (*req*, *resp_obj*)

show (*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

3.7.140 The nova.api.openstack.compute.plugins.v3.extension_info Module

class ExtensionInfo(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Extension information.

alias = 'extensions'

get_controller_extensions()

get_resources()

name = 'Extensions'

version = 1

class ExtensionInfoController(*extension_info*)
Bases: nova.api.openstack.wsgi.Controller

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class FakeExtension(*name*, *alias*)
Bases: object

3.7.141 The nova.api.openstack.compute.plugins.v3.fixed_ips Module

class FixedIPController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

reserve(**args*, ***kwargs*)

show(**args*, ***kwargs*)
Return data about the given fixed ip.

unreserve(**args*, ***kwargs*)

versioned_methods = {'_fill_reserved_status': [<nova.api.openstack.versioned_method.VersionedMethod object at 0x

wsgi_actions = {'unreserve': 'unreserve', 'reserve': 'reserve'}

wsgi_extensions = []

class FixedIps(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Fixed IPs support.

alias = 'os-fixed-ips'

get_controller_extensions()

get_resources()

name = 'FixedIPs'

version = 1

3.7.142 The nova.api.openstack.compute.plugins.v3.flavor_access Module

The flavor access extension.

class FlavorAccess(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Flavor access support.

alias = 'os-flavor-access'

get_controller_extensions()

get_resources()

name = 'FlavorAccess'

version = 1

class FlavorAccessController
Bases: nova.api.openstack.wsgi.Controller

The flavor access API controller for the OpenStack API.

index(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class FlavorActionController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

The flavor access API controller for the OpenStack API.

create(*req*, *body*, *resp_obj*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {'addTenantAccess': '_add_tenant_access', 'removeTenantAccess': '_remove_tenant_access'}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

3.7.143 The nova.api.openstack.compute.plugins.v3.flavor_manage Module

class FlavorManage(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Flavor create/delete API support.

alias = 'os-flavor-manage'

get_controller_extensions()

get_resources()

name = 'FlavorManage'

version = 1

class FlavorManageController
Bases: nova.api.openstack.wsgi.Controller

The Flavor Lifecycle API controller for the OpenStack API.

wsgi_actions = {'create': '_create', 'delete': '_delete'}

wsgi_extensions = []

3.7.144 The nova.api.openstack.compute.plugins.v3.flavor_rxtx Module

The Flavor Rxtx API extension.

class FlavorRxtx(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Support to show the rxtx status of a flavor.

alias = 'os-flavor-rxtx'

get_controller_extensions()

get_resources()

name = 'FlavorRxtx'

version = 1

class FlavorRxtxController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', 'create'), ('detail', None)]

3.7.145 The nova.api.openstack.compute.plugins.v3.flavors Module

class Flavors(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Flavors Extension.

alias = 'flavors'

get_controller_extensions()

get_resources()

name = 'Flavors'

version = 1

class `FlavorsController`(*view_builder=None*)
Bases: `nova.api.openstack.wsgi.Controller`

Flavor controller for the OpenStack API.

`detail`(**args*, ***kwargs*)
Return all flavors in detail.

`index`(**args*, ***kwargs*)
Return all flavors in brief.

`show`(**args*, ***kwargs*)
Return data about the given flavor id.

`wsgi_actions` = {}

`wsgi_extensions` = []

3.7.146 The `nova.api.openstack.compute.plugins.v3.flavors_extraspecs` Module

class `FlavorExtraSpecsController`(**args*, ***kwargs*)
Bases: `nova.api.openstack.wsgi.Controller`

The flavor extra specs API controller for the OpenStack API.

`create`(**args*, ***kwargs*)

`delete`(**args*, ***kwargs*)
Deletes an existing extra spec.

`index`(**args*, ***kwargs*)
Returns the list of extra specs for a given flavor.

`show`(**args*, ***kwargs*)
Return a single extra spec item.

`update`(**args*, ***kwargs*)

`wsgi_actions` = {}

`wsgi_extensions` = []

class `FlavorsExtraSpecs`(*extension_info*)
Bases: `nova.api.openstack.extensions.V3APIExtensionBase`

Flavors extra specs support.

`alias` = 'os-flavor-extra-specs'

`get_controller_extensions`()

`get_resources`()

`name` = 'FlavorExtraSpecs'

`version` = 1

3.7.147 The nova.api.openstack.compute.plugins.v3.floating_ip_dns Module

class FloatingIPDNSDomainController
Bases: nova.api.openstack.wsgi.Controller

DNS domain controller for OpenStack API.

delete(**args*, ***kwargs*)
Delete the domain identified by id.

index(**args*, ***kwargs*)
Return a list of available DNS domains.

update(**args*, ***kwargs*)
Add or modify domain entry.

wsgi_actions = {}

wsgi_extensions = []

class FloatingIPDNSEntryController
Bases: nova.api.openstack.wsgi.Controller

DNS Entry controller for OpenStack API.

delete(**args*, ***kwargs*)
Delete the entry identified by req and id.

show(**args*, ***kwargs*)
Return the DNS entry that corresponds to domain_id and id.

update(**args*, ***kwargs*)
Add or modify dns entry.

wsgi_actions = {}

wsgi_extensions = []

class FloatingIpDns(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Floating IP DNS support.

alias = 'os-floating-ip-dns'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'FloatingIpDns'

version = 1

3.7.148 The nova.api.openstack.compute.plugins.v3.floating_ip_pools Module

class FloatingIPPoolsController
Bases: nova.api.openstack.wsgi.Controller

The Floating IP Pool API controller for the OpenStack API.

index(**args*, ***kwargs*)
Return a list of pools.

wsgi_actions = {}

wsgi_extensions = []

class **FloatingIpPools**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Floating IPs support.

alias = 'os-floating-ip-pools'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'FloatingIpPools'

version = 1

3.7.149 The nova.api.openstack.compute.plugins.v3.floating_ips Module

class **FloatingIPActionController**(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'removeFloatingIp': '_remove_floating_ip', 'addFloatingIp': '_add_floating_ip'}

wsgi_extensions = []

class **FloatingIPController**
Bases: object

The Floating IPs API controller for the OpenStack API.

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)
Return a list of floating ips allocated to a project.

show(**args*, ***kwargs*)
Return data about the given floating ip.

class **FloatingIps**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Floating IPs support.

alias = 'os-floating-ips'

get_controller_extensions()

get_resources()

name = 'FloatingIps'

version = 1

disassociate_floating_ip(*self*, *context*, *instance*, *address*)

get_instance_by_floating_ip_addr(*self*, *context*, *address*)

3.7.150 The nova.api.openstack.compute.plugins.v3.floating_ips_bulk Module

class FloatingIPBulkController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

create(**args*, ***kwargs*)
Bulk create floating ips.

index(**args*, ***kwargs*)
Return a list of all floating ips.

show(**args*, ***kwargs*)
Return a list of all floating ips for a given host.

update(**args*, ***kwargs*)
Bulk delete floating IPs.

wsgi_actions = {}

wsgi_extensions = []

class FloatingIpsBulk(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Bulk handling of Floating IPs.

alias = 'os-floating-ips-bulk'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'FloatingIpsBulk'

version = 1

3.7.151 The nova.api.openstack.compute.plugins.v3.fping Module

class Fping(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Fping Management Extension.

alias = 'os-fping'

get_controller_extensions()

get_resources()

name = 'Fping'

version = 1

class FpingController(*network_api=None*)
Bases: nova.api.openstack.wsgi.Controller

check_fping()

static fping(*ips*)

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.152 The nova.api.openstack.compute.plugins.v3.hide_server_addresses Module

Extension for hiding server addresses in certain states.

class Controller(**args*, ***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

class HideServerAddresses(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Support hiding server addresses in certain states.

alias = 'os-hide-server-addresses'

get_controller_extensions()

get_resources()

name = 'HideServerAddresses'

version = 1

3.7.153 The nova.api.openstack.compute.plugins.v3.hosts Module

The hosts admin extension.

class HostController

Bases: nova.api.openstack.wsgi.Controller

The Hosts API controller for the OpenStack API.

index(**args*, ***kwargs*)

Returns a dict in the format

{'hosts': [{'host_name': 'some.host.name',
'service': 'cells',
'zone': 'internal'},
{'host_name': 'some.other.host.name',
'service': 'cells',
'zone': 'internal'},
{'host_name': 'some.celly.host.name',
'service': 'cells',
'zone': 'internal'},
{'host_name': 'console1.host.com',

'service': 'consoleauth',
'zone': 'internal'},
{'host_name': 'network1.host.com',
'service': 'network',
'zone': 'internal'},
{'host_name': 'netwwork2.host.com',
'service': 'network',
'zone': 'internal'},
{'host_name': 'compute1.host.com',
'service': 'compute',
'zone': 'nova'},
{'host_name': 'compute2.host.com',
'service': 'compute',
'zone': 'nova'},
{'host_name': 'sched1.host.com',
'service': 'scheduler',
'zone': 'internal'},
{'host_name': 'sched2.host.com',
'service': 'scheduler',
'zone': 'internal'},
{'host_name': 'vol1.host.com',
'service': 'volume',
'zone': 'internal'}]}

reboot(**args*, ***kwargs*)

show(**args*, ***kwargs*)
Shows the physical/usage resource given by hosts.

Parameters **id** – hostname

Returns

expected to use HostShowTemplate. ex.:

```
{'host': {'resource':D},..}
D: {'host': 'hostname','project': 'admin',
    'cpu': 1, 'memory_mb': 2048, 'disk_gb': 30}
```

shutdown(**args*, ***kwargs*)

startup(**args*, ***kwargs*)

update(**args*, ***kwargs*)
Return booleanized version of body dict.

Parameters

- **req** (*Request*) – The request object (containing 'nova-context' env var).
- **id** (*str*) – The host name.
- **body** (*dict*) – example format {'host': {'status': 'enable', 'maintenance_mode': 'enable'}}

Returns Same dict as body but 'enable' strings for 'status' and 'maintenance_mode' are converted into True, else False.

Return type dict

wsgi_actions = {}

wsgi_extensions = []

class Hosts(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin-only host administration.

alias = 'os-hosts'

get_controller_extensions()

get_resources()

name = 'Hosts'

version = 1

3.7.154 The nova.api.openstack.compute.plugins.v3.hypervisors Module

The hypervisors admin extension.

class Hypervisors(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin-only hypervisor administration.

alias = 'os-hypervisors'

get_controller_extensions()

get_resources()

name = 'Hypervisors'

version = 1

class HypervisorsController

Bases: nova.api.openstack.wsgi.Controller

The Hypervisors API controller for the OpenStack API.

detail(**args*, ***kwargs*)

index(**args*, ***kwargs*)

search(**args*, ***kwargs*)

servers(**args*, ***kwargs*)

show(**args*, ***kwargs*)

statistics(**args*, ***kwargs*)

uptime(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.155 The nova.api.openstack.compute.plugins.v3.image_metadata Module

class ImageMetadata(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Image Metadata API.

alias = 'image-metadata'

get_controller_extensions()

get_resources()

image_metadata_map(*mapper*, *wsgi_resource*)

name = 'ImageMetadata'

version = 1

class ImageMetadataController
Bases: nova.api.openstack.wsgi.Controller

The image metadata API controller for the OpenStack API.

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)
Returns the list of metadata for a given instance.

show(**args*, ***kwargs*)

update(**args*, ***kwargs*)

update_all(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.156 The nova.api.openstack.compute.plugins.v3.image_size Module

class ImageSize(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Adds image size to image listings.

alias = 'image-size'

get_controller_extensions()

get_resources()

name = 'ImageSize'

version = 1

class ImageSizeController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

3.7.157 The nova.api.openstack.compute.plugins.v3.images Module

class Images(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Proxying API for Images.

alias = 'images'

get_controller_extensions()

get_resources()

name = 'Images'

version = 1

class ImagesController(***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

Base controller for retrieving/displaying images.

delete(**args*, ***kwargs*)
Delete an image, if allowed.

Parameters

- **req** – *wsgi.Request* object
- **id** – Image identifier (integer)

detail(**args*, ***kwargs*)
Return a detailed index listing of images available to the request.

Parameters **req** – *wsgi.Request* object.

index(**args*, ***kwargs*)
Return an index listing of images available to the request.

Parameters **req** – *wsgi.Request* object

show(**args*, ***kwargs*)
Return detailed information about a specific image.

Parameters

- **req** – *wsgi.Request* object
- **id** – Image identifier

wsgi_actions = {}

wsgi_extensions = []

3.7.158 The nova.api.openstack.compute.plugins.v3.instance_actions Module

class InstanceActions(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

View a log of actions and events taken on an instance.

alias = 'os-instance-actions'

get_controller_extensions()
It's an abstract function V3APIExtensionBase and the extension will not be loaded without it.

get_resources()

name = 'InstanceActions'

version = 1

class InstanceActionsController
Bases: nova.api.openstack.wsgi.Controller

index(**args*, ***kwargs*)
Returns the list of actions recorded for a given instance.

show(**args*, ***kwargs*)
Return data about the given instance action.

wsgi_actions = {}

wsgi_extensions = []

3.7.159 The nova.api.openstack.compute.plugins.v3.instance_usage_audit_log Module

class InstanceUsageAuditLog(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin-only Task Log Monitoring.

alias = 'os-instance-usage-audit-log'

get_controller_extensions()

get_resources()

name = 'OSInstanceUsageAuditLog'

version = 1

class InstanceUsageAuditLogController
Bases: nova.api.openstack.wsgi.Controller

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.160 The nova.api.openstack.compute.plugins.v3.ips Module

class IPs(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Server addresses.

alias = 'ips'

get_controller_extensions()

get_resources()

name = 'Ips'

version = 1

class IPsController(***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

The servers addresses API controller for the OpenStack API.

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.161 The nova.api.openstack.compute.plugins.v3.keypairs Module

Keypair management extension.

class Controller(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('detail', None)]

class KeypairController
Bases: nova.api.openstack.wsgi.Controller

Keypair API controller for the OpenStack API.

create(**args*, ***kwargs*)
Create or import keypair.

Sending name will generate a key and return private_key and fingerprint.

You can send a public_key to add an existing ssh key.

params: keypair object with: name (required) - string public_key (optional) - string

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

versioned_methods = {'index': [<nova.api.openstack.versioned_method.VersionedMethod object at 0x7f85896c20d0>

wsgi_actions = {}

wsgi_extensions = []

class Keypairs(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Keypair Support.

alias = 'os-keypairs'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'Keypairs'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

3.7.162 The nova.api.openstack.compute.plugins.v3.limits Module

class Limits(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Limits support.

alias = 'limits'

get_controller_extensions()

get_resources()

name = 'Limits'

version = 1

class LimitsController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

Controller for accessing limits in the OpenStack API.

index(**args*, ***kwargs*)
Return all global and rate limit information.

wsgi_actions = {}

wsgi_extensions = []

3.7.163 The nova.api.openstack.compute.plugins.v3.lock_server Module

class LockServer(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enable lock/unlock server actions.

alias = 'os-lock-server'

get_controller_extensions()

get_resources()

name = 'LockServer'

version = 1

class LockServerController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'lock': '_lock', 'unlock': '_unlock'}

wsgi_extensions = []

3.7.164 The nova.api.openstack.compute.plugins.v3.migrate_server Module

class MigrateServer(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enable migrate and live-migrate server actions.

alias = 'os-migrate-server'

get_controller_extensions()

get_resources()

name = 'MigrateServer'

version = 1

class MigrateServerController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'migrate': '_migrate', 'os-migrateLive': '_migrate_live'}

wsgi_extensions = []

3.7.165 The nova.api.openstack.compute.plugins.v3.migrations Module

class Migrations(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Provide data on migrations.

alias = 'os-migrations'

get_controller_extensions()

get_resources()

name = 'Migrations'

version = 1

class MigrationsController
Bases: nova.api.openstack.wsgi.Controller

Controller for accessing migrations in OpenStack API.

index(**args*, ***kwargs*)
Return all migrations in progress.

wsgi_actions = {}

wsgi_extensions = []

authorize(*context*, *action_name*)

output(*migrations_obj*)
Returns the desired output of the API from an object.

From a MigrationsList's object this method returns a list of primitive objects with the only necessary fields.

3.7.166 The nova.api.openstack.compute.plugins.v3.multinic Module

The multinic extension.

class Multinic(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Multiple network support.

alias = 'os-multinic'

get_controller_extensions()

get_resources()

name = 'Multinic'

version = 1

class MultinicController(**args*, ***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'addFixedIp': '_add_fixed_ip', 'removeFixedIp': '_remove_fixed_ip'}

wsgi_extensions = []

3.7.167 The nova.api.openstack.compute.plugins.v3.multiple_create Module

class MultipleCreate(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Allow multiple create in the Create Server v3 API.

alias = 'os-multiple-create'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'MultipleCreate'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

3.7.168 The nova.api.openstack.compute.plugins.v3.networks Module

class NetworkController(*network_api=None*)

Bases: nova.api.openstack.wsgi.Controller

add(**args*, ***kwargs*)

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {'disassociate': '_disassociate_host_and_project'}

wsgi_extensions = []

class Networks (*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Admin-only Network Management Extension.

alias = 'os-networks'

get_controller_extensions ()

get_resources ()

name = 'Networks'

version = 1

network_dict (*context*, *network*)

3.7.169 The nova.api.openstack.compute.plugins.v3.networks_associate Module

class NetworkAssociateActionController (*network_api=None*)
Bases: nova.api.openstack.wsgi.Controller

Network Association API Controller.

wsgi_actions = {'disassociate_project': '_disassociate_project_only', 'associate_host': '_associate_host', 'disassociate_

wsgi_extensions = []

class NetworksAssociate (*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Network association support.

alias = 'os-networks-associate'

get_controller_extensions ()

get_resources ()

name = 'NetworkAssociationSupport'

version = 1

3.7.170 The nova.api.openstack.compute.plugins.v3.pause_server Module

class PauseServer (*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enable pause/unpause server actions.

alias = 'os-pause-server'

get_controller_extensions ()

get_resources ()

name = 'PauseServer'

version = 1

class `PauseServerController`(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

`wsgi_actions` = {'pause': '_pause', 'unpause': '_unpause'}

`wsgi_extensions` = []

3.7.171 The nova.api.openstack.compute.plugins.v3.pci Module

class `Pci`(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Pci access support.

`alias` = 'os-pci'

`get_controller_extensions`()

`get_resources`()

`name` = 'PciAccess'

`version` = 1

class `PciController`
Bases: nova.api.openstack.wsgi.Controller

`detail`(**args*, ***kwargs*)

`index`(**args*, ***kwargs*)

`show`(**args*, ***kwargs*)

`wsgi_actions` = {}

`wsgi_extensions` = []

class `PciHypervisorController`(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

`detail`(*req*, *resp_obj*)

`show`(*req*, *resp_obj*, *id*)

`wsgi_actions` = {}

`wsgi_extensions` = [('show', None), ('detail', None)]

class `PciServerController`(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

`detail`(*req*, *resp_obj*)

`show`(*req*, *resp_obj*, *id*)

`wsgi_actions` = {}

`wsgi_extensions` = [('detail', None), ('show', None)]

3.7.172 The nova.api.openstack.compute.plugins.v3.personality Module

class `Personality`(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Personality support.

alias = 'os-personality'

get_controller_extensions()

get_resources()

get_server_create_schema()

get_server_rebuild_schema()

name = 'Personality'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

server_rebuild(*server_dict*, *create_kwargs*, *body_deprecated_param=None*)

version = 1

3.7.173 The nova.api.openstack.compute.plugins.v3.preserve_ephemeral_rebuild Module

class PreserveEphemeralRebuild(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Allow preservation of the ephemeral partition on rebuild.

alias = 'os-preserve-ephemeral-rebuild'

get_controller_extensions()

get_resources()

get_server_rebuild_schema()

name = 'PreserveEphemeralOnRebuild'

server_rebuild(*rebuild_dict*, *rebuild_kwargs*, *body_deprecated_param=None*)

version = 1

3.7.174 The nova.api.openstack.compute.plugins.v3.quota_classes Module

class QuotaClassSetsController(***kwargs*)

Bases: nova.api.openstack.wsgi.Controller

show(**args*, ***kwargs*)

supported_quotas = []

update(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class QuotaClasses(*extension_info*)

Bases: nova.api.openstack.extensions.V3APIExtensionBase

Quota classes management support.

alias = 'os-quota-class-sets'

get_controller_extensions()

get_resources()

name = 'QuotaClasses'

version = 1

3.7.175 The nova.api.openstack.compute.plugins.v3.quota_sets Module

class QuotaSets(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Quotas management support.

alias = 'os-quota-sets'

get_controller_extensions()

get_resources()

name = 'Quotas'

version = 1

class QuotaSetsController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

defaults(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

detail(**args*, ***kwargs*)

show(**args*, ***kwargs*)

update(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.176 The nova.api.openstack.compute.plugins.v3.remote_consoles Module

class RemoteConsoles(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Interactive Console support.

alias = 'os-remote-consoles'

get_controller_extensions()

get_resources()

name = 'Consoles'

version = 1

class RemoteConsolesController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

create(**args*, ***kwargs*)

get_rdp_console(**args*, ***kwargs*)
Get text console output.

get_serial_console(**args*, ***kwargs*)
Get connection to a serial console.

get_spice_console(**args*, ***kwargs*)
Get text console output.

get_vnc_console(**args*, ***kwargs*)
Get text console output.

versioned_methods = {'get_spice_console': [<nova.api.openstack.versioned_method.VersionedMethod object at 0x7f8

wsgi_actions = {'os-getRDPConsole': 'get_rdp_console', 'os-getSPICEConsole': 'get_spice_console', 'os-getSerialCon

wsgi_extensions = []

3.7.177 The nova.api.openstack.compute.plugins.v3.rescue Module

The rescue mode extension.

class Rescue(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Instance rescue mode.

alias = 'os-rescue'

get_controller_extensions()

get_resources()

name = 'Rescue'

version = 1

class RescueController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'unrescue': '_unrescue', 'rescue': '_rescue'}

wsgi_extensions = []

3.7.178 The nova.api.openstack.compute.plugins.v3.scheduler_hints Module

class SchedulerHints(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Pass arbitrary key/value pairs to the scheduler.

alias = 'os-scheduler-hints'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'SchedulerHints'

server_create(*server_dict*, *create_kwargs*, *req_body*)

version = 1

3.7.179 The nova.api.openstack.compute.plugins.v3.security_group_default_rules Module

class SecurityGroupDefaultRules(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Default rules for security group support.

alias = 'os-security-group-default-rules'

get_controller_extensions()

get_resources()

name = 'SecurityGroupDefaultRules'

version = 1

class SecurityGroupDefaultRulesController
Bases: nova.api.openstack.compute.plugins.v3.security_groups.SecurityGroupControllerBase

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.180 The nova.api.openstack.compute.plugins.v3.security_groups Module

The security groups extension.

class SecurityGroupActionController(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'removeSecurityGroup': '_removeSecurityGroup', 'addSecurityGroup': '_addSecurityGroup'}

wsgi_extensions = []

class SecurityGroupController
Bases: nova.api.openstack.compute.plugins.v3.security_groups.SecurityGroupControllerBase

The Security group API controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates a new security group.

delete(**args*, ***kwargs*)
Delete a security group.

index(**args*, ***kwargs*)
Returns a list of security groups.

show(**args*, ***kwargs*)
Return data about the given security group.

update(**args*, ***kwargs*)
Update a security group.

wsgi_actions = {}

wsgi_extensions = []

class **SecurityGroupControllerBase**
Bases: nova.api.openstack.wsgi.Controller

Base class for Security Group controllers.

wsgi_actions = {}

wsgi_extensions = []

class **SecurityGroupRulesController**
Bases: nova.api.openstack.compute.plugins.v3.security_groups.SecurityGroupControllerBase

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class **SecurityGroups**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Security group support.

alias = 'os-security-groups'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'SecurityGroups'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

class **SecurityGroupsOutputController**(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

create(*req*, *resp_obj*, *body*)

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('create', None), ('detail', None)]

class **ServerSecurityGroupController**
Bases: nova.api.openstack.compute.plugins.v3.security_groups.SecurityGroupControllerBase

index(**args*, ***kwargs*)
Returns a list of security groups for the given instance.

wsgi_actions = {}

wsgi_extensions = []

3.7.181 The nova.api.openstack.compute.plugins.v3.server_diagnostics Module

class ServerDiagnostics(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Allow Admins to view server diagnostics through server action.

alias = 'os-server-diagnostics'

get_controller_extensions()

get_resources()

name = 'ServerDiagnostics'

version = 1

class ServerDiagnosticsController
Bases: nova.api.openstack.wsgi.Controller

index(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.182 The nova.api.openstack.compute.plugins.v3.server_external_events Module

class ServerExternalEvents(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Server External Event Triggers.

alias = 'os-server-external-events'

get_controller_extensions()

get_resources()

name = 'ServerExternalEvents'

version = 1

class ServerExternalEventsController
Bases: nova.api.openstack.wsgi.Controller

create(**args*, ***kwargs*)
Creates a new instance event.

wsgi_actions = {}

wsgi_extensions = []

3.7.183 The nova.api.openstack.compute.plugins.v3.server_groups Module

The Server Group API Extension.

class ServerGroupController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

The Server group API controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates a new server group.

delete(**args*, ***kwargs*)
Delete an server group.

index(**args*, ***kwargs*)
Returns a list of server groups.

show(**args*, ***kwargs*)
Return data about the given server group.

wsgi_actions = {}

wsgi_extensions = []

class ServerGroups(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Server group support.

alias = 'os-server-groups'

get_controller_extensions()

get_resources()

name = 'ServerGroups'

version = 1

3.7.184 The nova.api.openstack.compute.plugins.v3.server_metadata Module

class ServerMetadata(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Server Metadata API.

alias = 'server-metadata'

get_controller_extensions()

get_resources()

name = 'ServerMetadata'

server_metadata_map(*mapper*, *wsgi_resource*)

version = 1

class ServerMetadataController
Bases: nova.api.openstack.wsgi.Controller

The server metadata API controller for the OpenStack API.

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)
Deletes an existing metadata.

index(**args*, ***kwargs*)
Returns the list of metadata for a given instance.

show(**args*, ***kwargs*)
Return a single metadata item.

update(**args*, ***kwargs*)

update_all(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.185 The nova.api.openstack.compute.plugins.v3.server_password Module

The server password extension.

class ServerPassword(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Server password support.

alias = 'os-server-password'

get_controller_extensions()

get_resources()

name = 'ServerPassword'

version = 1

class ServerPasswordController
Bases: nova.api.openstack.wsgi.Controller

The Server Password API controller for the OpenStack API.

clear(**args*, ***kwargs*)
Removes the encrypted server password from the metadata server

Note that this does not actually change the instance server password.

index(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.186 The nova.api.openstack.compute.plugins.v3.server_usage Module

class ServerUsage(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Adds launched_at and terminated_at on Servers.

alias = 'os-server-usage'

get_controller_extensions()

get_resources()

name = 'ServerUsage'

version = 1

class **ServerUsageController**(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('detail', None), ('show', None)]

3.7.187 The nova.api.openstack.compute.plugins.v3.servers Module

class **Servers**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Servers.

alias = 'servers'

get_controller_extensions()

get_resources()

name = 'Servers'

version = 1

class **ServersController**(***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

The Server API base controller class for the OpenStack API.

B64_REGEX = <_sre.SRE_Pattern object at 0x7f85899dfcf0>

EXTENSION_CREATE_NAMESPACE = 'nova.api.v3.extensions.server.create'

EXTENSION_DESERIALIZE_EXTRACT_REBUILD_NAMESPACE = 'nova.api.v3.extensions.server.rebuild.deserialize'

EXTENSION_DESERIALIZE_EXTRACT_SERVER_NAMESPACE = 'nova.api.v3.extensions.server.create.deserialize'

EXTENSION_REBUILD_NAMESPACE = 'nova.api.v3.extensions.server.rebuild'

EXTENSION_RESIZE_NAMESPACE = 'nova.api.v3.extensions.server.resize'

EXTENSION_UPDATE_NAMESPACE = 'nova.api.v3.extensions.server.update'

create(**args*, ***kwargs*)
Creates a new server for a given user.

delete(**args*, ***kwargs*)
Destroys a server.

detail(**args*, ***kwargs*)
Returns a list of server details for a given user.

index (**args*, ***kwargs*)
Returns a list of server names and ids for a given user.

schema_server_create = {'additionalProperties': False, 'required': ['server'], 'type': 'object', 'properties': {'server

schema_server_rebuild = {'additionalProperties': False, 'required': ['rebuild'], 'type': 'object', 'properties': {'rebu

schema_server_resize = {'additionalProperties': False, 'required': ['resize'], 'type': 'object', 'properties': {'resize':

schema_server_update = {'additionalProperties': False, 'required': ['server'], 'type': 'object', 'properties': {'server

show (**args*, ***kwargs*)
Returns server details by server id.

update (**args*, ***kwargs*)
Update server then pass on to version-specific controller.

wsgi_actions = {'createImage': '_action_create_image', 'rebuild': '_action_rebuild', 'reboot': '_action_reboot', 'os-st

wsgi_extensions = []

remove_invalid_options (*context*, *search_options*, *allowed_search_options*)
Remove search options that are not valid for non-admin API/context.

3.7.188 The nova.api.openstack.compute.plugins.v3.services Module

class ServiceController
Bases: nova.api.openstack.wsgi.Controller

delete (**args*, ***kwargs*)
Deletes the specified service.

index (**args*, ***kwargs*)
Return a list of all running services. Filter by host & service name

update (**args*, ***kwargs*)
Perform service update

wsgi_actions = {}

wsgi_extensions = []

class Services (*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Services support.

alias = 'os-services'

get_controller_extensions ()

get_resources ()

name = 'Services'

version = 1

3.7.189 The nova.api.openstack.compute.plugins.v3.shelve Module

The shelved mode extension.

class **Shelve**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Instance shelve mode.

alias = 'os-shelve'

get_controller_extensions()

get_resources()

name = 'Shelve'

version = 1

class **ShelveController**(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'unshelve': '_unshelve', 'shelveOffload': '_shelve_offload', 'shelve': '_shelve'}

wsgi_extensions = []

3.7.190 The nova.api.openstack.compute.plugins.v3.simple_tenant_usage Module

class **SimpleTenantUsage**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Simple tenant usage extension.

alias = 'os-simple-tenant-usage'

get_controller_extensions()

get_resources()

name = 'SimpleTenantUsage'

version = 1

class **SimpleTenantUsageController**(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

index(**args*, ***kwargs*)
Retrieve tenant_usage for all tenants.

show(**args*, ***kwargs*)
Retrieve tenant_usage for a specified tenant.

wsgi_actions = {}

wsgi_extensions = []

parse_strtime(*dstr*, *fmt*)

3.7.191 The nova.api.openstack.compute.plugins.v3.suspend_server Module

class **SuspendServer**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Enable suspend/resume server actions.

alias = 'os-suspend-server'

get_controller_extensions()

get_resources()

name = 'SuspendServer'

version = 1

class **SuspendServerController**(**args*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

wsgi_actions = {'suspend': '_suspend', 'resume': '_resume'}

wsgi_extensions = []

3.7.192 The nova.api.openstack.compute.plugins.v3.tenant_networks Module

class **TenantNetworkController**(*network_api=None*)
Bases: nova.api.openstack.wsgi.Controller

create(**args*, ***kwargs*)

delete(**args*, ***kwargs*)

index(**args*, ***kwargs*)

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class **TenantNetworks**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Tenant-based Network Management Extension.

alias = 'os-tenant-networks'

get_controller_extensions()

get_resources()

name = 'OSTenantNetworks'

version = 1

network_dict(*network*)

3.7.193 The nova.api.openstack.compute.plugins.v3.used_limits Module

class **UsedLimits**(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Provide data on limited resources that are being used.

alias = 'os-used-limits'

get_controller_extensions()

get_resources()

name = 'UsedLimits'

version = 1

class UsedLimitsController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

index(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = [('index', None)]

3.7.194 The nova.api.openstack.compute.plugins.v3.user_data Module

class UserData(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Add user_data to the Create Server API.

alias = 'os-user-data'

get_controller_extensions()

get_resources()

get_server_create_schema()

name = 'UserData'

server_create(*server_dict*, *create_kwargs*, *body_deprecated_param*)

version = 1

3.7.195 The nova.api.openstack.compute.plugins.v3.versions Module

class Versions(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

API Version information.

alias = 'versions'

get_controller_extensions()

get_resources()

name = 'Versions'

version = 1

version_map(*mapper*, *wsgi_resource*)

class VersionsController(*view_builder=None*)
Bases: nova.api.openstack.wsgi.Controller

show(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

3.7.196 The nova.api.openstack.compute.plugins.v3.virtual_interfaces Module

The virtual interfaces extension.

class ServerVirtualInterfaceController
Bases: nova.api.openstack.wsgi.Controller

The instance VIF API controller for the OpenStack API.

index(**args*, ***kwargs*)
Returns the list of VIFs for a given instance.

wsgi_actions = {}

wsgi_extensions = []

class VirtualInterfaces(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Virtual interface support.

alias = 'os-virtual-interfaces'

get_controller_extensions()

get_resources()

name = 'VirtualInterfaces'

version = 1

3.7.197 The nova.api.openstack.compute.plugins.v3.volumes Module

The volumes extension.

class SnapshotController
Bases: nova.api.openstack.wsgi.Controller

The Snapshots API controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates a new snapshot.

delete(**args*, ***kwargs*)
Delete a snapshot.

detail(**args*, ***kwargs*)
Returns a detailed list of snapshots.

index(**args*, ***kwargs*)
Returns a summary list of snapshots.

show(**args*, ***kwargs*)
Return data about the given snapshot.

wsgi_actions = {}

wsgi_extensions = []

class VolumeAttachmentController
Bases: nova.api.openstack.wsgi.Controller

The volume attachment API controller for the OpenStack API.

A child resource of the server. Note that we use the volume id as the ID of the attachment (though this is not guaranteed externally)

create(**args*, ***kwargs*)
Attach a volume to an instance.

delete(**args*, ***kwargs*)
Detach a volume from an instance.

index(**args*, ***kwargs*)
Returns the list of volume attachments for a given instance.

show(**args*, ***kwargs*)
Return data about the given volume attachment.

update(**args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class VolumeController
Bases: nova.api.openstack.wsgi.Controller

The Volumes API controller for the OpenStack API.

create(**args*, ***kwargs*)
Creates a new volume.

delete(**args*, ***kwargs*)
Delete a volume.

detail(**args*, ***kwargs*)
Returns a detailed list of volumes.

index(**args*, ***kwargs*)
Returns a summary list of volumes.

show(**args*, ***kwargs*)
Return data about the given volume.

wsgi_actions = {}

wsgi_extensions = []

class Volumes(*extension_info*)
Bases: nova.api.openstack.extensions.V3APIExtensionBase

Volumes support.

alias = 'os-volumes'

get_controller_extensions()

get_resources()

name = 'Volumes'

version = 1

3.7.198 The nova.api.openstack.compute.schemas.v3.access_ips Module

3.7.199 The nova.api.openstack.compute.schemas.v3.admin_password Module

3.7.200 The nova.api.openstack.compute.schemas.v3.agents Module

3.7.201 The nova.api.openstack.compute.schemas.v3.aggregates Module

3.7.202 The nova.api.openstack.compute.schemas.v3.assisted_volume_snapshots Module

3.7.203 The nova.api.openstack.compute.schemas.v3.attach_interfaces Module

3.7.204 The nova.api.openstack.compute.schemas.v3.availability_zone Module

3.7.205 The nova.api.openstack.compute.schemas.v3.block_device_mapping Module

3.7.206 The nova.api.openstack.compute.schemas.v3.block_device_mapping_v1 Module

3.7.207 The nova.api.openstack.compute.schemas.v3.cells Module

3.7.208 The nova.api.openstack.compute.schemas.v3.cloudpipe Module

3.7.209 The nova.api.openstack.compute.schemas.v3.config_drive Module

3.7.210 The nova.api.openstack.compute.schemas.v3.console_output Module

3.7.211 The nova.api.openstack.compute.schemas.v3.create_backup Module

3.7.212 The nova.api.openstack.compute.schemas.v3.disk_config Module

3.7.213 The nova.api.openstack.compute.schemas.v3.evacuate Module

3.7.214 The nova.api.openstack.compute.schemas.v3.fixed_ips Module

3.7.215 The nova.api.openstack.compute.schemas.v3.flavor_access Module

3.7.216 The nova.api.openstack.compute.schemas.v3.flavor_manage Module

3.7.217 The nova.api.openstack.compute.schemas.v3.flavors_extraspecs Module

3.7.218 The nova.api.openstack.compute.schemas.v3.floating_ip_dns Module

The server metadata API controller for the OpenStack API.

create(*req*, *server_id*, *body*)

delete(*req*, *server_id*, *id*)
Deletes an existing metadata.

index(*req*, *server_id*)
Returns the list of metadata for a given instance.

show(*req*, *server_id*, *id*)
Return a single metadata item.

update(*req*, *server_id*, *id*, *body*)

update_all(*req*, *server_id*, *body*)

create_resource()

3.7.247 The nova.api.openstack.compute.servers Module

class Controller(*ext_mgr=None*, ***kwargs*)
Bases: nova.api.openstack.wsgi.Controller

The Server API base controller class for the OpenStack API.

B64_REGEX = <_sre.SRE_Pattern object at 0x7f85899dfcf0>

create(*req*, *body*)
Creates a new server for a given user.

delete(*req*, *id*)
Destroys a server.

detail(*req*)
Returns a list of server details for a given user.

index(*req*)
Returns a list of server names and ids for a given user.

show(*req*, *id*)
Returns server details by server id.

update(*req*, *id*, *body*)
Update server then pass on to version-specific controller.

wsgi_actions = {'createImage': '_action_create_image', 'rebuild': '_action_rebuild', 'reboot': '_action_reboot', 'chan

wsgi_extensions = []

create_resource(*ext_mgr*)

remove_invalid_options(*context*, *search_options*, *allowed_search_options*)
Remove search options that are not valid for non-admin API/context.

3.7.248 The nova.api.openstack.compute.versions Module

class VersionV2
Bases: object

show(*req*)

class Versions
Bases: nova.api.openstack.wsgi.Resource

get_action_args(*request_environment*)
Parse dictionary created by routes library.

index(*req*, *body=None*)
Return all versions.

multi(*req*, *body=None*)
Return multiple choices.

create_resource()

3.7.249 The nova.api.openstack.compute.views.addresses Module

class ViewBuilder
Bases: nova.api.openstack.common.ViewBuilder

Models server addresses as a dictionary.

basic(*ip*, *extend_address=False*)
Return a dictionary describing an IP address.

index(*networks*, *extend_address=False*)
Return a dictionary describing a list of networks.

show(*network*, *label*, *extend_address=False*)
Returns a dictionary describing a network.

class ViewBuilderV3
Bases: nova.api.openstack.compute.views.addresses.ViewBuilder

Models server addresses as a dictionary.

basic(*ip*, *extend_address=False*)
Return a dictionary describing an IP address.

3.7.250 The nova.api.openstack.compute.views.flavors Module

class V3ViewBuilder
Bases: nova.api.openstack.compute.views.flavors.ViewBuilder

show(*request*, *flavor*)

class ViewBuilder
Bases: nova.api.openstack.common.ViewBuilder

basic(*request*, *flavor*)

detail(*request*, *flavors*)
Return the 'detail' view of flavors.

index(*request*, *flavors*)
Return the 'index' view of flavors.

show(*request*, *flavor*)

3.7.251 The nova.api.openstack.compute.views.images Module

class ViewBuilder
Bases: nova.api.openstack.common.ViewBuilder

basic(*request*, *image*)
Return a dictionary with basic image attributes.

detail(*request*, *images*)
Show a list of images with details.

index(*request*, *images*)
Show a list of images with basic attributes.

show(*request*, *image*)
Return a dictionary with image details.

3.7.252 The nova.api.openstack.compute.views.limits Module

class ViewBuilder
Bases: object

OpenStack API base limits view builder.

build(*rate_limits*, *absolute_limits*)

limit_names = {}

class ViewBuilderV3
Bases: nova.api.openstack.compute.views.limits.ViewBuilder

3.7.253 The nova.api.openstack.compute.views.servers Module

class ViewBuilder
Bases: nova.api.openstack.common.ViewBuilder

Model a server API response as a python dictionary.

basic(*request*, *instance*)
Generic, non-detailed view of an instance.

create(*request*, *instance*)
View that should be returned when an instance is created.

detail(*request*, *instances*)
Detailed view of a list of instance.

index(*request*, *instances*)
Show a list of servers without many details.

show(*request*, *instance*)
Detailed view of a single instance.

class ViewBuilderV3
Bases: nova.api.openstack.compute.views.servers.ViewBuilder

Model a server V3 API response as a python dictionary.

show(*request*, *instance*, *extend_address=True*)
Detailed view of a single instance.

3.7.254 The nova.api.openstack.compute.views.versions Module

class ViewBuilder(*base_url*)
Bases: nova.api.openstack.common.ViewBuilder

build_choices(*VERSIONS*, *req*)

build_version(*version*)

build_versions(*versions*)

generate_href(*version*, *path=None*)
Create an url that refers to a specific version_number.

get_view_builder(*req*)

3.7.255 The nova.api.openstack.extensions Module

class ControllerExtension(*extension*, *collection*, *controller*)
Bases: object

Extend core controllers of nova OpenStack API.

Provide a way to extend existing nova OpenStack API core controllers.

class ExtensionDescriptor(*ext_mgr*)
Bases: object

Base class that defines the contract for extensions.

Note that you don't have to derive from this class to have a valid extension; it is purely a convenience.

alias = None

get_controller_extensions()
List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

get_resources()
List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

is_valid()
Validate required fields for extensions.

Raises an attribute error if the attr is not defined

name = None

updated = None

class ExtensionManager
Bases: object

Load extensions from the configured extension path.

See nova/tests/api/openstack/compute/extensions/foxinsocks.py or an example extension implementation.

get_controller_extensions()
Returns a list of ControllerExtension objects.

get_resources()
Returns a list of ResourceExtension objects.

is_loaded(*alias*)

load_extension(*ext_factory*)
Execute an extension factory.

Loads an extension. The 'ext_factory' is the name of a callable that will be imported and called with one argument–the extension manager. The factory callable is expected to call the register() method at least once.

register(*ext*)

sorted_extensions()

class **ExtensionsController**(*extension_manager*)
Bases: nova.api.openstack.wsgi.Resource

create(*req*, *body*)

delete(*req*, *id*)

index(*req*)

show(*req*, *id*)

class **ResourceExtension**(*collection*, *controller=None*, *parent=None*, *collection_actions=None*, *member_actions=None*, *custom_routes_fn=None*, *inherits=None*, *member_name=None*)
Bases: object

Add top level resources to the OpenStack API in nova.

class **V3APIExtensionBase**(*extension_info*)
Bases: object

Abstract base class for all V3 API extensions.

All V3 API extensions must derive from this class and implement the abstract methods get_resources and get_controller_extensions even if they just return an empty list. The extensions must also define the abstract properties.

alias
Alias for the extension.

get_controller_extensions()
Return a list of controller extensions.

The extensions should return a list of ControllerExtension objects. This list may be empty.

get_resources()
Return a list of resources extensions.

The extensions should return a list of ResourceExtension objects. This list may be empty.

is_valid()
Validate required fields for extensions.

Raises an attribute error if the attr is not defined

name
Name of the extension.

version
Version of the extension.

check_compute_policy(*context*, *action*, *target*, *scope='compute'*)

core_authorizer(*api_name*, *extension_name*)

expected_errors(*errors*)
Decorator for v3 API methods which specifies expected exceptions.

Specify which exceptions may occur when an API method is called. If an unexpected exception occurs then return a 500 instead and ask the user of the API to file a bug report.

extension_authorizer(*api_name*, *extension_name*)

load_standard_extensions(*ext_mgr*, *logger*, *path*, *package*, *ext_list=None*)
Registers all standard API extensions.

os_compute_authorizer(*extension_name*)

os_compute_soft_authorizer(*extension_name*)

soft_core_authorizer(*api_name*, *extension_name*)

soft_extension_authorizer(*api_name*, *extension_name*)

3.7.256 The nova.api.openstack.urlmap Module

class **Accept**(*value*)
Bases: object

best_match(*supported_content_types*)

class **URLMap**(*not_found_app=None*)
Bases: paste.urlmap.URLMap

parse_list_header(*value*)
Parse lists as described by RFC 2068 Section 2.

In particular, parse comma-separated lists where the elements of the list may include quoted-strings. A quoted-string could contain a comma. A non-quoted string could have quotes in the middle. Quotes are removed automatically after parsing.

The return value is a standard list:

```
>>> parse_list_header('token, "quoted value"')
['token', 'quoted value']
```

Parameters **value** – a string with a list header.

Returns list

parse_options_header(*value*)
Parse a Content-Type like header into a tuple with the content type and the options:

```
>>> parse_options_header('Content-Type: text/html; mimetype=text/html')
('Content-Type:', {'mimetype': 'text/html'})
```

Parameters **value** – the header to parse.

Returns (str, options)

unquote_header_value(*value*)
Unquotes a header value. This does not use the real unquoting but what browsers are actually using for quoting.

Parameters **value** – the header value to unquote.

urlmap_factory(*loader*, *global_conf*, ***local_conf*)

3.7.257 The nova.api.openstack.versioned_method Module

class VersionedMethod(*name*, *start_version*, *end_version*, *func*)
Bases: object

3.7.258 The nova.api.openstack.wsgi Module

class ActionDispatcher
Bases: object

Maps method name to local methods through action name.

default(*data*)

dispatch(**args*, ***kwargs*)
Find and call local method.

class Controller(*view_builder=None*)
Bases: object

Default controller.

classmethod api_version(*min_ver*, *max_ver=None*)
Decorator for versioning api methods.

Add the decorator to any method which takes a request object as the first parameter and belongs to a class which inherits from wsgi.Controller.

@min_ver: string representing minimum version @max_ver: optional string representing maximum version

static is_valid_body(*body*, *entity_name*)

wsgi_actions = {}

wsgi_extensions = []

class ControllerMetaclass
Bases: type

Controller metaclass.

This metaclass automates the task of assembling a dictionary mapping action keys to method names.

class DictSerializer
Bases: nova.api.openstack.wsgi.ActionDispatcher

Default request body serialization.

default(*data*)

serialize(*data*, *action='default'*)

exception Fault(*exception*)
Bases: webob.exc.HTTPException

Wrap webob.exc.HTTPException to provide API friendly response.

class JSONDeserializer
Bases: nova.api.openstack.wsgi.TextDeserializer

default(*datastring*)

class JSONDictSerializer
Bases: nova.api.openstack.wsgi.DictSerializer

Default JSON request body serialization.

default(*data*)

exception RateLimitFault(*message*, *details*, *retry_time*)
Bases: webob.exc.HTTPException

Rate-limited request response.

class Request(**args*, ***kwargs*)
Bases: webob.request.Request

Add some OpenStack API-specific logic to the base webob.Request.

best_match_content_type()
Determine the requested response content-type.

best_match_language()
Determine the best available language for the request.

Returns the best language match or None if the 'Accept-Language' header was not available in the request.

cache_db_compute_node(*compute_node*)

cache_db_compute_nodes(*compute_nodes*)

cache_db_flavor(*flavor*)

cache_db_flavors(*flavors*)

cache_db_instance(*instance*)

cache_db_instances(*instances*)

cache_db_items(*key*, *items*, *item_key='id'*)
Allow API methods to store objects from a DB query to be used by API extensions within the same API request.

An instance of this class only lives for the lifetime of a single API request, so there's no need to implement full cache management.

get_content_type()
Determine content type of the request body.

Does not do any body introspection, only checks header

get_db_compute_node(*id*)

get_db_compute_nodes()

get_db_flavor(*flavorid*)

get_db_flavors()

get_db_instance(*instance_uuid*)

get_db_instances()

get_db_item(*key*, *item_key*)
Allow an API extension to get a previously stored object within the same API request.

Note that the object data will be slightly stale.

get_db_items(*key*)
Allow an API extension to get previously stored objects within the same API request.

Note that the object data will be slightly stale.

set_api_version_request()
Set API version request based on the request header information.

class Resource(*controller*, *action_peek=None*, *inherits=None*, ***deserializers*)
Bases: nova.wsgi.Application

WSGI app that handles (de)serialization and controller dispatch.

WSGI app that reads routing information supplied by RoutesMiddleware and calls the requested action method upon its controller. All controller action methods must accept a 'req' argument, which is the incoming wsgi.Request. If the operation is a PUT or POST, the controller method must also accept a 'body' argument (the deserialized request body). They may raise a webob.exc exception or return a dict, which will be serialized by requested content type.

Exceptions derived from webob.exc.HTTPException will be automatically wrapped in Fault() to provide API friendly error responses.

deserialize(*meth*, *content_type*, *body*)

dispatch(*method*, *request*, *action_args*)
Dispatch a call to the action-specific method.

get_action_args(*request_environment*)
Parse dictionary created by routes library.

get_body(*request*)

get_method(*request*, *action*, *content_type*, *body*)

post_process_extensions(*extensions*, *resp_obj*, *request*, *action_args*)

pre_process_extensions(*extensions*, *request*, *action_args*)

register_actions(*controller*)
Registers controller actions with this resource.

register_extensions(*controller*)
Registers controller extensions with this resource.

support_api_request_version = False

class ResourceExceptionHandler
Bases: object

Context manager to handle Resource exceptions.

Used when processing exceptions generated by API implementation methods (or their extensions). Converts most exceptions to Fault exceptions, with the appropriate logging.

class ResourceV21(*controller*, *action_peek=None*, *inherits=None*, ***deserializers*)
Bases: nova.api.openstack.wsgi.Resource

support_api_request_version = True

class ResponseObject(*obj*, *code=None*, *headers=None*, ***serializers*)
Bases: object

Bundles a response object with appropriate serializers.

Object that app methods may return in order to bind alternate serializers with a response object to be serialized. Its use is optional.

attach (***kwargs*)
Attach slave templates to serializers.

code
Retrieve the response status.

get_serializer (*content_type*, *default_serializers=None*)
Returns the serializer for the wrapped object.

Returns the serializer for the wrapped object subject to the indicated content type. If no serializer matching the content type is attached, an appropriate serializer drawn from the default serializers will be used. If no appropriate serializer is available, raises InvalidContentType.

headers
Retrieve the headers.

preserialize (*content_type*, *default_serializers=None*)
Prepares the serializer that will be used to serialize.

Determines the serializer that will be used and prepares an instance of it for later call. This allows the serializer to be accessed by extensions for, e.g., template extension.

serialize (*request*, *content_type*, *default_serializers=None*)
Serializes the wrapped object.

Utility method for serializing the wrapped object. Returns a webob.Response object.

class **TextDeserializer**
Bases: nova.api.openstack.wsgi.ActionDispatcher

Default request body deserialization.

default (*datastring*)

deserialize (*datastring*, *action='default'*)

action (*name*)
Mark a function as an action.

The given name will be taken as the action key in the body.

This is also overloaded to allow extensions to provide non-extending definitions of create and delete operations.

action_peek_json (*body*)
Determine action to invoke.

deserializers (***deserializers*)
Attaches deserializers to a method.

This decorator associates a dictionary of deserializers with a method. Note that the function attributes are directly manipulated; the method is not wrapped.

extends (**args*, ***kwargs*)
Indicate a function extends an operation.

Can be used as either:

```
@extends
def index(...):
    pass
```

or as:

```
@extends(action='resize')
def _action_resize(...):
    pass
```

get_media_map()

get_supported_content_types()

response(*code*)
Attaches response code to a method.

This decorator associates a response code with a method. Note that the function attributes are directly manipulated; the method is not wrapped.

serializers(*******serializers*)
Attaches serializers to a method.

This decorator associates a dictionary of serializers with a method. Note that the function attributes are directly manipulated; the method is not wrapped.

3.7.259 The nova.api.opts Module

list_opts()

3.7.260 The nova.api.sizelimit Module

Request Body limiting middleware.

3.7.261 The nova.api.validation.parameter_types Module

Common parameter types for validating request Body.

3.7.262 The nova.api.validation.validators Module

Internal implementation of request Body validating middleware.

3.7.263 The nova.api.validator Module

validate(*args*, *validator*)
Validate values of args against validators in validator.

Parameters

- **args** – Dict of values to be validated.
- **validator** – A dict where the keys map to keys in args and the values are validators. Applies each validator to `args[key]`

Returns True if validation succeeds. Otherwise False.

A validator should be a callable which accepts 1 argument and which returns True if the argument passes validation. False otherwise. A validator should not raise an exception to indicate validity of the argument.

Only validates keys which show up in both args and validator.

validate_image_path(*val*)

validate_int(*max_value=None*)

validate_str(*max_length=None*)

validate_url_path(*val*)
True if val is matched by the path component grammar in rfc3986.

validate_user_data(*user_data*)
Check if the user_data is encoded properly.

3.7.264 The nova.availability_zones Module

Availability zone helper functions.

get_availability_zones(*context*, *get_only_available=False*, *with_hosts=False*)
Return available and unavailable zones on demand.

Parameters

- **get_only_available** – flag to determine whether to return available zones only, default False indicates return both available zones and not available zones, True indicates return available zones only
- **with_hosts** (*bool*) – whether to return hosts part of the AZs

get_host_availability_zone(*context*, *host*)

get_instance_availability_zone(*context*, *instance*)
Return availability zone of specified instance.

reset_cache()
Reset the cache, mainly for testing purposes and update availability_zone for host aggregate

set_availability_zones(*context*, *services*)

update_host_availability_zone_cache(*context*, *host*, *availability_zone=None*)

3.7.265 The nova.baserpc Module

Base RPC client and server common to all services.

class BaseAPI(*topic*)
Bases: object

Client side of the base rpc API.

API version history:

1.0 - Initial version. 1.1 - Add get_backdoor_port

VERSION_ALIASES = {}

get_backdoor_port(*context*, *host*)

ping(*context*, *arg*, *timeout=None*)

class BaseRPCAPI(*service_name*, *backdoor_port*)
Bases: object

Server side of the base RPC API.

get_backdoor_port(*context*)

ping(*context*, *arg*)

target = <Target namespace=baseapi, version=1.1>

3.7.266 The nova.block_device Module

class BlockDeviceDict(*bdm_dict=None*, *do_not_default=None*, ***kwargs*)
Bases: dict

Represents a Block Device Mapping in Nova.

classmethod from_api(*api_dict*, *image_uuid_specified*)
Transform the API format of data to the internally used one.

Only validate if the source_type field makes sense.

classmethod from_legacy(*legacy_bdm*)

get_image_mapping()

legacy()

create_blank_bdm(*size*, *guest_format=None*)

create_image_bdm(*image_ref*, *boot_index=0*)
Create a block device dict based on the image_ref.

This is useful in the API layer to keep the compatibility with having an image_ref as a field in the instance requests

ephemeral_num(*ephemeral_name*)

from_legacy_mapping(*legacy_block_device_mapping*, *image_uuid=''*, *root_device_name=None*, *no_root=False*)
Transform a legacy list of block devices to the new data format.

get_bdm_ephemeral_disk_size(*block_device_mappings*)

get_bdm_local_disk_num(*block_device_mappings*)

get_bdm_swap_list(*block_device_mappings*)

get_bdms_to_connect(*bdms*, *exclude_root_mapping=False*)
Will return non-root mappings, when exclude_root_mapping is true. Otherwise all mappings will be returned.

get_device_letter(*device_name*)

get_root_bdm(*bdms*)

instance_block_mapping(*instance*, *bdms*)

is_ephemeral(*device_name*)

is_safe_for_update(*block_device_dict*)
Determine if passed dict is a safe subset for update.

Safe subset in this case means a safe subset of both legacy and new versions of data, that can be passed to an UPDATE query without any transformation.

is_swap_or_ephemeral(*device_name*)

legacy_mapping(*block_device_mapping*)
Transform a list of block devices of an instance back to the legacy data format.

mappings_prepend_dev(*mappings*)
Prepend '/dev/' to 'device' entry of swap/ephemeral virtual type.

match_device(*device*)
Matches device name and returns prefix, suffix.

new_format_is_ephemeral(*bdm*)

new_format_is_swap(*bdm*)

prepend_dev(*device_name*)
Make sure there is a leading '/dev/'.

properties_root_device_name(*properties*)
get root device name from image meta data. If it isn't specified, return None.

snapshot_from_bdm(*snapshot_id*, *template*)
Create a basic volume snapshot BDM from a given template bdm.

strip_dev(*device_name*)
remove leading '/dev/'.

strip_prefix(*device_name*)
remove both leading /dev/ and xvd or sd or vd or hd.

validate_and_default_volume_size(*bdm*)

validate_device_name(*value*)

volume_in_mapping(*mount_device*, *block_device_info*)

3.7.267 The nova.cells.driver Module

Base Cells Communication Driver

class BaseCellsDriver
Bases: object

The base class for cells communication.

One instance of this class will be created for every neighbor cell that we find in the DB and it will be associated with the cell in its CellState.

One instance is also created by the cells manager for setting up the consumers.

send_message_to_cell(*cell_state*, *message*)
Send a message to a cell.

start_servers(*msg_runner*)
Start any messaging servers the driver may need.

stop_servers()
Stop accepting messages.

3.7.268 The nova.cells.filters.different_cell Module

Different cell filter.

A scheduler hint of 'different_cell' with a value of a full cell name may be specified to route a build away from a particular cell.

class DifferentCellFilter
Bases: nova.cells.filters.BaseCellFilter

Different cell filter. Works by specifying a scheduler hint of 'different_cell'. The value should be the full cell path.

filter_all (*cells*, *filter_properties*)
Override filter_all() which operates on the full list of cells...

3.7.269 The nova.cells.filters.image_properties Module

Image properties filter.

Image metadata named 'hypervisor_version_requires' with a version specification may be specified to ensure the build goes to a cell which has hypervisors of the required version.

If either the version requirement on the image or the hypervisor capability of the cell is not present, this filter returns without filtering out the cells.

class ImagePropertiesFilter
Bases: nova.cells.filters.BaseCellFilter

Image properties filter. Works by specifying the hypervisor required in the image metadata and the supported hypervisor version in cell capabilities.

filter_all (*cells*, *filter_properties*)
Override filter_all() which operates on the full list of cells...

3.7.270 The nova.cells.filters.target_cell Module

Target cell filter.

A scheduler hint of 'target_cell' with a value of a full cell name may be specified to route a build to a particular cell. No error handling is done as there's no way to know whether the full path is a valid.

class TargetCellFilter
Bases: nova.cells.filters.BaseCellFilter

Target cell filter. Works by specifying a scheduler hint of 'target_cell'. The value should be the full cell path.

filter_all (*cells*, *filter_properties*)
Override filter_all() which operates on the full list of cells...

3.7.271 The nova.cells.manager Module

Cells Service Manager

class CellsManager (**args*, ***kwargs*)
Bases: nova.manager.Manager

The nova-cells manager class. This class defines RPC methods that the local cell may call. This class is NOT used for messages coming from other cells. That communication is driver-specific.

Communication to other cells happens via the nova.cells.messaging module. The MessageRunner from that module will handle routing the message to the correct cell via the communications driver. Most methods below create 'targeted' (where we want to route a message to a specific cell) or 'broadcast' (where we want a message to go to multiple cells) messages.

Scheduling requests get passed to the scheduler class.

action_events_get (*ctxt*, *cell_name*, *action_id*)

action_get_by_request_id(*ctxt*, *cell_name*, *instance_uuid*, *request_id*)

actions_get(*ctxt*, *cell_name*, *instance_uuid*)

backup_instance(*ctxt*, *instance*, *image_id*, *backup_type*, *rotation*)
Backup an instance in its cell.

bdm_destroy_at_top(*ctxt*, *instance_uuid*, *device_name=None*, *volume_id=None*)
BDM was destroyed for instance in this cell. Tell the API cells.

bdm_update_or_create_at_top(*ctxt*, *bdm*, *create=None*)
BDM was created/updated in this cell. Tell the API cells.

build_instances(*ctxt*, *build_inst_kwargs*)
Pick a cell (possibly ourselves) to build new instance(s) and forward the request accordingly.

bw_usage_update_at_top(*ctxt*, *bw_update_info*)
Update bandwidth usage at top level cell.

cell_create(*ctxt*, *values*)

cell_delete(*ctxt*, *cell_name*)

cell_get(*ctxt*, *cell_name*)

cell_update(*ctxt*, *cell_name*, *values*)

compute_node_get(**args*, ***kwargs*)

compute_node_get_all(*ctxt*, *hypervisor_match=None*)
Return list of compute nodes in all cells.

compute_node_stats(*ctxt*)
Return compute node stats totals from all cells.

confirm_resize(*ctxt*, *instance*)
Confirm a resize for an instance in its cell.

consoleauth_delete_tokens(*ctxt*, *instance_uuid*)
Delete consoleauth tokens for an instance in API cells.

get_capacities(*ctxt*, *cell_name*)

get_cell_info_for_neighbors(*_ctxt*)
Return cell information for our neighbor cells.

get_host_uptime(*ctxt*, *host_name*)
Return host uptime for a compute host in a certain cell

Parameters **host_name** – fully qualified hostname. It should be in format of parent!child@host_id

get_migrations(*ctxt*, *filters*)
Fetch migrations applying the filters.

inject_network_info(*ctxt*, *instance*)
Inject networking for an instance in its cell.

instance_delete_everywhere(*ctxt*, *instance*, *delete_type*)
This is used by API cell when it didn't know what cell an instance was in, but the instance was requested to be deleted or soft_deleted. So, we'll broadcast this everywhere.

instance_destroy_at_top(*ctxt*, *instance*)
Destroy an instance at the top level cell.

instance_fault_create_at_top(*ctxt*, *instance_fault*)
Create an instance fault at the top level cell.

instance_update_at_top(*ctxt*, *instance*)
Update an instance at the top level cell.

instance_update_from_api(*ctxt*, *instance*, *expected_vm_state*, *expected_task_state*, *admin_state_reset*)
Update an instance in its cell.

live_migrate_instance(*ctxt*, *instance*, *block_migration*, *disk_over_commit*, *host_name*)
Live migrate an instance in its cell.

pause_instance(*ctxt*, *instance*)
Pause an instance in its cell.

post_start_hook()
Have the driver start its servers for inter-cell communication. Also ask our child cells for their capacities and capabilities so we get them more quickly than just waiting for the next periodic update. Receiving the updates from the children will cause us to update our parents. If we don't have any children, just update our parents immediately.

proxy_rpc_to_manager(**args*, ***kwargs*)

reboot_instance(*ctxt*, *instance*, *reboot_type*)
Reboot an instance in its cell.

rebuild_instance(*ctxt*, *instance*, *image_href*, *admin_password*, *files_to_inject*, *preserve_ephemeral*, *kwargs*)

reset_network(*ctxt*, *instance*)
Reset networking for an instance in its cell.

resize_instance(*ctxt*, *instance*, *flavor*, *extra_instance_updates*, *clean_shutdown=True*)
Resize an instance in its cell.

resume_instance(*ctxt*, *instance*)
Resume an instance in its cell.

revert_resize(*ctxt*, *instance*)
Revert a resize for an instance in its cell.

run_compute_api_method(*ctxt*, *cell_name*, *method_info*, *call*)
Call a compute API method in a specific cell.

service_delete(*ctxt*, *cell_service_id*)
Deletes the specified service.

service_get_all(*ctxt*, *filters*)
Return services in this cell and in all child cells.

service_get_by_compute_host(**args*, ***kwargs*)

service_update(*ctxt*, *host_name*, *binary*, *params_to_update*)
Used to enable/disable a service. For compute services, setting to disabled stops new builds arriving on that host.

Parameters

- **host_name** – the name of the host machine that the service is running
- **binary** – The name of the executable that the service runs as
- **params_to_update** – eg. {'disabled': True}

Returns the service reference

set_admin_password(*ctxt*, *instance*, *new_pass*)

snapshot_instance(*ctxt*, *instance*, *image_id*)
Snapshot an instance in its cell.

soft_delete_instance(*ctxt*, *instance*)
Soft-delete an instance in its cell.

start_instance(*ctxt*, *instance*)
Start an instance in its cell.

stop_instance(*ctxt*, *instance*, *do_cast=True*, *clean_shutdown=True*)
Stop an instance in its cell.

suspend_instance(*ctxt*, *instance*)
Suspend an instance in its cell.

sync_instances(*ctxt*, *project_id*, *updated_since*, *deleted*)
Force a sync of all instances, potentially by project_id, and potentially since a certain date/time.

target = <Target version=1.35>

task_log_get_all(*ctxt*, *task_name*, *period_beginning*, *period_ending*, *host=None*, *state=None*)
Get task logs from the DB from all cells or a particular cell.

If 'host' is not None, host will be of the format 'cell!name@host', with '@host' being optional. The query will be directed to the appropriate cell and return all task logs, or task logs matching the host if specified.

'state' also may be None. If it's not, filter by the state as well.

terminate_instance(*ctxt*, *instance*)
Delete an instance in its cell.

unpause_instance(*ctxt*, *instance*)
Unpause an instance in its cell.

validate_console_port(*ctxt*, *instance_uuid*, *console_port*, *console_type*)
Validate console port with child cell compute node.

3.7.272 The nova.cells.messaging Module

Cell messaging module.

This module defines the different message types that are passed between cells and the methods that they can call when the target cell has been reached.

The interface into this module is the MessageRunner class.

class MessageRunner(*state_manager*)
Bases: object

This class is the main interface into creating messages and processing them.

Public methods in this class are typically called by the CellsManager to create a new message and process it with the exception of 'message_from_json' which should be used by CellsDrivers to convert a JSONified message it has received back into the appropriate Message class.

Private methods are used internally when we need to keep some 'global' state. For instance, eventlet queues used for responses are held in this class. Also, when a Message is process()ed above and it's determined we should take action locally, _process_message_locally() will be called.

When needing to add a new method to call in a Cell2Cell message, define the new method below and also add it to the appropriate MessageMethods class where the real work will be done.

action_events_get(*ctxt*, *cell_name*, *action_id*)

action_get_by_request_id(*ctxt*, *cell_name*, *instance_uuid*, *request_id*)

actions_get(*ctxt*, *cell_name*, *instance_uuid*)

ask_children_for_capabilities(*ctxt*)
Tell child cells to send us capabilities. This is typically called on startup of the nova-cells service.

ask_children_for_capacities(*ctxt*)
Tell child cells to send us capacities. This is typically called on startup of the nova-cells service.

backup_instance(*ctxt*, *instance*, *image_id*, *backup_type*, *rotation*)
Backup an instance in its cell.

bdm_destroy_at_top(*ctxt*, *instance_uuid*, *device_name=None*, *volume_id=None*)
Destroy a BDM at top level cell.

bdm_update_or_create_at_top(*ctxt*, *bdm*, *create=None*)
Update/Create a BDM at top level cell.

build_instances(*ctxt*, *target_cell*, *build_inst_kwargs*)
Called by the cell scheduler to tell a child cell to build instance(s).

bw_usage_update_at_top(*ctxt*, *bw_update_info*)
Update bandwidth usage at top level cell.

compute_node_get(*ctxt*, *cell_name*, *compute_id*)
Return compute node entry from a specific cell by ID.

compute_node_get_all(*ctxt*, *hypervisor_match=None*)
Return list of compute nodes in all child cells.

compute_node_stats(*ctxt*)
Return compute node stats from all child cells.

confirm_resize(*ctxt*, *instance*)
Confirm a resize for an instance in its cell.

consoleauth_delete_tokens(*ctxt*, *instance_uuid*)
Delete consoleauth tokens for an instance in API cells.

get_host_uptime(*ctxt*, *cell_name*, *host_name*)

static get_message_types()

get_migrations(*ctxt*, *cell_name*, *run_locally*, *filters*)
Fetch all migrations applying the filters for a given cell or all cells.

inject_network_info(*ctxt*, *instance*)
Inject networking for an instance in its cell.

instance_delete_everywhere(*ctxt*, *instance*, *delete_type*)
This is used by API cell when it didn't know what cell an instance was in, but the instance was requested to be deleted or soft_deleted. So, we'll broadcast this everywhere.

instance_destroy_at_top(*ctxt*, *instance*)
Destroy an instance at the top level cell.

instance_fault_create_at_top(*ctxt*, *instance_fault*)
Create an instance fault at the top level cell.

instance_update_at_top(*ctxt*, *instance*)
Update an instance at the top level cell.

instance_update_from_api(*ctxt*, *instance*, *expected_vm_state*, *expected_task_state*, *admin_state_reset*)
Update an instance object in its cell.

live_migrate_instance(*ctxt*, *instance*, *block_migration*, *disk_over_commit*, *host_name*)
Live migrate an instance in its cell.

message_from_json(*json_message*)
Turns a message in JSON format into an appropriate Message instance. This is called when cells receive a message from another cell.

pause_instance(*ctxt*, *instance*)
Pause an instance in its cell.

proxy_rpc_to_manager(*ctxt*, *cell_name*, *host_name*, *topic*, *rpc_message*, *call*, *timeout*)

reboot_instance(*ctxt*, *instance*, *reboot_type*)
Reboot an instance in its cell.

rebuild_instance(*ctxt*, *instance*, *image_href*, *admin_password*, *files_to_inject*, *preserve_ephemeral*, *kwargs*)

reset_network(*ctxt*, *instance*)
Reset networking for an instance in its cell.

resize_instance(*ctxt*, *instance*, *flavor*, *extra_instance_updates*, *clean_shutdown=True*)
Resize an instance in its cell.

resume_instance(*ctxt*, *instance*)
Resume an instance in its cell.

revert_resize(*ctxt*, *instance*)
Revert a resize for an instance in its cell.

run_compute_api_method(*ctxt*, *cell_name*, *method_info*, *call*)
Call a compute API method in a specific cell.

service_delete(*ctxt*, *cell_name*, *service_id*)
Deletes the specified service.

service_get_all(*ctxt*, *filters=None*)

service_get_by_compute_host(*ctxt*, *cell_name*, *host_name*)

service_update(*ctxt*, *cell_name*, *host_name*, *binary*, *params_to_update*)
Used to enable/disable a service. For compute services, setting to disabled stops new builds arriving on that host.

Parameters

- **host_name** – the name of the host machine that the service is running
- **binary** – The name of the executable that the service runs as
- **params_to_update** – eg. {'disabled': True}

Returns the update service object

set_admin_password(*ctxt*, *instance*, *new_pass*)

snapshot_instance(*ctxt*, *instance*, *image_id*)
Snapshot an instance in its cell.

soft_delete_instance(*ctxt*, *instance*)

start_instance(*ctxt*, *instance*)
Start an instance in its cell.

stop_instance(*ctxt*, *instance*, *do_cast=True*, *clean_shutdown=True*)
Stop an instance in its cell.

suspend_instance(*ctxt*, *instance*)
Suspend an instance in its cell.

sync_instances(*ctxt*, *project_id*, *updated_since*, *deleted*)
Force a sync of all instances, potentially by project_id, and potentially since a certain date/time.

task_log_get_all(*ctxt*, *cell_name*, *task_name*, *period_beginning*, *period_ending*, *host=None*, *state=None*)
Get task logs from the DB from all cells or a particular cell.

If 'cell_name' is None or '', get responses from all cells. If 'host' is not None, filter by host. If 'state' is not None, filter by state.

Return a list of Response objects.

tell_parents_our_capabilities(*ctxt*)
Send our capabilities to parent cells.

tell_parents_our_capacities(*ctxt*)
Send our capacities to parent cells.

terminate_instance(*ctxt*, *instance*)

unpause_instance(*ctxt*, *instance*)
Unpause an instance in its cell.

validate_console_port(*ctxt*, *cell_name*, *instance_uuid*, *console_port*, *console_type*)
Validate console port with child cell compute node.

class Response(*ctxt*, *cell_name*, *value*, *failure*)
Bases: `object`

Holds a response from a cell. If there was a failure, 'failure' will be True and 'response' will contain an encoded Exception.

classmethod from_json(*ctxt*, *json_message*)

to_json()

value_or_raise()

deserialize_remote_exception(*data*, *allowed_remote_exmods*)

serialize_remote_exception(*failure_info*, *log_failure=True*)
Prepares exception data to be sent over rpc.

Failure_info should be a sys.exc_info() tuple.

3.7.273 The nova.cells.opts Module

Global cells config options

get_cell_type()
Return the cell type, 'api', 'compute', or None (if cells is disabled).

list_opts()

3.7.274 The nova.cells.rpc_driver Module

Cells RPC Communication Driver

class CellsRPCDriver(**args*, ***kwargs*)
Bases: nova.cells.driver.BaseCellsDriver

Driver for cell<->cell communication via RPC. This is used to setup the RPC consumers as well as to send a message to another cell.

One instance of this class will be created for every neighbor cell that we find in the DB and it will be associated with the cell in its CellState.

One instance is also created by the cells manager for setting up the consumers.

send_message_to_cell(*cell_state*, *message*)
Use the IntercellRPCAPI to send a message to a cell.

start_servers(*msg_runner*)
Start RPC servers.

Start up 2 separate servers for handling inter-cell communication via RPC. Both handle the same types of messages, but requests/replies are separated to solve potential deadlocks. (If we used the same queue for both, it's possible to exhaust the RPC thread pool while we wait for replies.. such that we'd never consume a reply.)

stop_servers()
Stop RPC servers.

NOTE: Currently there's no hooks when stopping services to have managers cleanup, so this is not currently called.

class InterCellRPCAPI
Bases: object

Client side of the Cell<->Cell RPC API.

The CellsRPCDriver uses this to make calls to another cell.

API version history: 1.0 - Initial version.

... Grizzly supports message version 1.0. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 1.0.

VERSION_ALIASES = {'grizzly': '1.0'}

send_message_to_cell(*cell_state*, *message*)
Send a message to another cell by JSON-ifying the message and making an RPC cast to 'process_message'. If the message says to fanout, do it. The topic that is used will be 'CONF.rpc_driver_queue_base.<message_type>'.

class InterCellRPCDispatcher(*msg_runner*)
Bases: object

RPC Dispatcher to handle messages received from other cells.

All messages received here have come from a sibling cell. Depending on the ultimate target and type of message, we may process the message in this cell, relay the message to another sibling cell, or both. This logic is defined by the message class in the nova.cells.messaging module.

process_message(*_ctxt*, *message*)
We received a message from another cell. Use the MessageRunner to turn this from JSON back into an instance of the correct Message class. Then process it!

target = <Target version=1.0>

3.7.275 The nova.cells.rpcapi Module

Client side of nova-cells RPC API (for talking to the nova-cells service within a cell).

This is different than communication between child and parent nova-cells services. That communication is handled by the cells driver via the messaging module.

class CellsAPI

Bases: object

Cells client-side RPC API

API version history:

- 1.0 - Initial version.
- 1.1 - Adds get_cell_info_for_neighbors() and sync_instances()
- **1.2 - Adds service_get_all(), service_get_by_compute_host(),** and proxy_rpc_to_compute_manager()
- 1.3 - Adds task_log_get_all()
- **1.4 - Adds compute_node_get(), compute_node_get_all(), and** compute_node_stats()
- **1.5 - Adds actions_get(), action_get_by_request_id(), and** action_events_get()
- 1.6 - Adds consoleauth_delete_tokens() and validate_console_port()

... Grizzly supports message version 1.6. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 1.6.

- 1.7 - Adds service_update()
- 1.8 - Adds build_instances(), deprecates schedule_run_instance()
- 1.9 - Adds get_capacities()
- 1.10 - Adds bdm_update_or_create_at_top(), and bdm_destroy_at_top()
- 1.11 - Adds get_migrations()
- 1.12 - Adds instance_start() and instance_stop()
- **1.13 - Adds cell_create(), cell_update(), cell_delete(), and** cell_get()
- 1.14 - Adds reboot_instance()
- 1.15 - Adds suspend_instance() and resume_instance()
- 1.16 - Adds instance_update_from_api()
- 1.17 - Adds get_host_uptime()
- 1.18 - Adds terminate_instance() and soft_delete_instance()
- 1.19 - Adds pause_instance() and unpause_instance()
- 1.20 - Adds resize_instance() and live_migrate_instance()
- 1.21 - Adds revert_resize() and confirm_resize()
- 1.22 - Adds reset_network()
- 1.23 - Adds inject_network_info()

- 1.24 - Adds backup_instance() and snapshot_instance()

... Havana supports message version 1.24. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.24.

- 1.25 - Adds rebuild_instance()
- 1.26 - Adds service_delete()
- 1.27 - Updates instance_delete_everywhere() for instance objects

... Icehouse supports message version 1.27. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.27.

- 1.28 - Make bdm_update_or_create_at_top and use bdm objects
- 1.29 - Adds set_admin_password()

... Juno supports message version 1.29. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.29.

- 1.30 - Make build_instances() use flavor object
- 1.31 - Add clean_shutdown to stop, resize, rescue, and shelve
- 1.32 - Send objects for instances in build_instances()
- 1.33 - Add clean_shutdown to resize_instance()
- **1.34 - build_instances uses BlockDeviceMapping objects, drops** legacy_bdm argument

... Kilo supports message version 1.34. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.34.

- **1.35 - Make instance_update_at_top, instance_destroy_at_top** and instance_info_cache_update_at_top use instance objects

VERSION_ALIASES = {'kilo': '1.34', 'grizzly': '1.6', 'havana': '1.24', 'juno': '1.29', 'icehouse': '1.27'}

action_events_get(*ctxt*, *instance*, *action_id*)

action_get_by_request_id(*ctxt*, *instance*, *request_id*)

actions_get(*ctxt*, *instance*)

backup_instance(*ctxt*, *instance*, *image_id*, *backup_type*, *rotation*)

bdm_destroy_at_top(*ctxt*, *instance_uuid*, *device_name=None*, *volume_id=None*)
Broadcast upwards that a block device mapping was destroyed. One of device_name or volume_id should be specified.

bdm_update_or_create_at_top(*ctxt*, *bdm*, *create=None*)
Create or update a block device mapping in API cells. If create is True, only try to create. If create is None, try to update but fall back to create. If create is False, only attempt to update. This maps to nova-conductor's behavior.

build_instances(*ctxt*, ***kwargs*)
Build instances.

bw_usage_update_at_top(*ctxt*, *uuid*, *mac*, *start_period*, *bw_in*, *bw_out*, *last_ctr_in*, *last_ctr_out*, *last_refreshed=None*)
Broadcast upwards that bw_usage was updated.

call_compute_api_method(*ctxt*, *cell_name*, *method*, **args*, ***kwargs*)
Make a call to a compute API method in a certain cell.

cast_compute_api_method(*ctxt*, *cell_name*, *method*, **args*, ***kwargs*)
Make a cast to a compute API method in a certain cell.

cell_create(*ctxt*, *values*)

cell_delete(*ctxt*, *cell_name*)

cell_get(*ctxt*, *cell_name*)

cell_update(*ctxt*, *cell_name*, *values*)

compute_node_get(*ctxt*, *compute_id*)
Get a compute node by ID in a specific cell.

compute_node_get_all(*ctxt*, *hypervisor_match=None*)
Return list of compute nodes in all cells, optionally filtering by hypervisor host.

compute_node_stats(*ctxt*)
Return compute node stats from all cells.

confirm_resize(*ctxt*, *instance*, *migration*, *host*, *reservations*, *cast=True*)

consoleauth_delete_tokens(*ctxt*, *instance_uuid*)
Delete consoleauth tokens for an instance in API cells.

get_capacities(*ctxt*, *cell_name=None*)

get_cell_info_for_neighbors(*ctxt*)
Get information about our neighbor cells from the manager.

get_host_uptime(*context*, *host_name*)
Gets the host uptime in a particular cell. The cell name should be encoded within the host_name

get_migrations(*ctxt*, *filters*)
Get all migrations applying the filters.

inject_network_info(*ctxt*, *instance*)
Inject networking for an instance.

instance_delete_everywhere(*ctxt*, *instance*, *delete_type*)
Delete instance everywhere. delete_type may be 'soft' or 'hard'. This is generally only used to resolve races when API cell doesn't know to what cell an instance belongs.

instance_destroy_at_top(*ctxt*, *instance*)
Destroy instance at API level.

instance_fault_create_at_top(*ctxt*, *instance_fault*)
Create an instance fault at the top.

instance_info_cache_update_at_top(*ctxt*, *instance_info_cache*)
Broadcast up that an instance's info_cache has changed.

instance_update_at_top(*ctxt*, *instance*)
Update instance at API level.

instance_update_from_api(*ctxt*, *instance*, *expected_vm_state*, *expected_task_state*, *admin_state_reset*)
Update an instance in its cell.

This method takes a new-world instance object.

live_migrate_instance(*ctxt*, *instance*, *host_name*, *block_migration*, *disk_over_commit*)

pause_instance(*ctxt*, *instance*)
Pause an instance in its cell.

This method takes a new-world instance object.

proxy_rpc_to_manager(*ctxt*, *rpc_message*, *topic*, *call=False*, *timeout=None*)
Proxy RPC to a compute manager. The host in the topic should be encoded with the target cell name.

reboot_instance(*ctxt*, *instance*, *block_device_info*, *reboot_type*)
Reboot an instance in its cell.

This method takes a new-world instance object.

rebuild_instance(*ctxt*, *instance*, *new_pass*, *injected_files*, *image_ref*, *orig_image_ref*, *orig_sys_metadata*, *bdms*, *recreate=False*, *on_shared_storage=False*, *host=None*, *preserve_ephemeral=False*, *kwargs=None*)

reset_network(*ctxt*, *instance*)
Reset networking for an instance.

resize_instance(*ctxt*, *instance*, *extra_instance_updates*, *scheduler_hint*, *flavor*, *reservations*, *clean_shutdown=True*)

resume_instance(*ctxt*, *instance*)
Resume an instance in its cell.

This method takes a new-world instance object.

revert_resize(*ctxt*, *instance*, *migration*, *host*, *reservations*)

service_delete(*ctxt*, *cell_service_id*)
Deletes the specified service.

service_get_all(*ctxt*, *filters=None*)
Ask all cells for their list of services.

service_get_by_compute_host(*ctxt*, *host_name*)
Get the service entry for a host in a particular cell. The cell name should be encoded within the host_name.

service_update(*ctxt*, *host_name*, *binary*, *params_to_update*)
Used to enable/disable a service. For compute services, setting to disabled stops new builds arriving on that host.

Parameters

- **host_name** – the name of the host machine that the service is running
- **binary** – The name of the executable that the service runs as
- **params_to_update** – eg. {'disabled': True}

set_admin_password(*ctxt*, *instance*, *new_pass*)

snapshot_instance(*ctxt*, *instance*, *image_id*)

soft_delete_instance(*ctxt*, *instance*, *reservations=None*)
Soft-delete an instance in its cell.

This method takes a new-world instance object.

start_instance(*ctxt*, *instance*)
Start an instance in its cell.

This method takes a new-world instance object.

stop_instance(*ctxt*, *instance*, *do_cast=True*, *clean_shutdown=True*)
Stop an instance in its cell.

This method takes a new-world instance object.

suspend_instance(*ctxt*, *instance*)
Suspend an instance in its cell.

This method takes a new-world instance object.

sync_instances(*ctxt*, *project_id=None*, *updated_since=None*, *deleted=False*)
Ask all cells to sync instance data.

task_log_get_all(*ctxt*, *task_name*, *period_beginning*, *period_ending*, *host=None*, *state=None*)
Get the task logs from the DB in child cells.

terminate_instance(*ctxt*, *instance*, *bdms*, *reservations=None*)
Delete an instance in its cell.

This method takes a new-world instance object.

unpause_instance(*ctxt*, *instance*)
Unpause an instance in its cell.

This method takes a new-world instance object.

validate_console_port(*ctxt*, *instance_uuid*, *console_port*, *console_type*)
Validate console port with child cell compute node.

3.7.276 The nova.cells.scheduler Module

Cells Scheduler

class CellsScheduler(*msg_runner*)
Bases: nova.db.base.Base

The cells scheduler.

build_instances(*message*, *build_inst_kwargs*)

3.7.277 The nova.cells.state Module

CellState Manager

class CellState(*cell_name*, *is_me=False*)
Bases: object

Holds information for a particular cell.

get_cell_info()
Return subset of cell information for OS API use.

send_message(*message*)
Send a message to a cell. Just forward this to the driver, passing ourselves and the message as arguments.

update_capabilities(*cell_metadata*)
Update cell capabilities for a cell.

update_capacities(*capacities*)
Update capacity information for a cell.

update_db_info(*cell_db_info*)
Update cell credentials from db.

class CellStateManager(*cell_state_cls=None*)
Bases: nova.db.base.Base

cell_get(**args*, ***kwargs*)

get_capacities(**args*, ***kwargs*)

get_cell_info_for_neighbors(**args*, ***kwargs*)
Return cell information for all neighbor cells.

get_child_cell(**args*, ***kwargs*)

get_child_cells(**args*, ***kwargs*)
Return list of child cell_infos.

get_my_state(**args*, ***kwargs*)
Return information for my (this) cell.

get_our_capabilities(**args*, ***kwargs*)

get_our_capacities(**args*, ***kwargs*)

get_parent_cell(**args*, ***kwargs*)

get_parent_cells(**args*, ***kwargs*)
Return list of parent cell_infos.

update_cell_capabilities(**args*, ***kwargs*)
Update capabilities for a cell.

update_cell_capacities(**args*, ***kwargs*)
Update capacities for a cell.

class CellStateManagerDB(*cell_state_cls=None*)
Bases: nova.cells.state.CellStateManager

cell_create(**args*, ***kwargs*)

cell_delete(**args*, ***kwargs*)

cell_update(**args*, ***kwargs*)

class CellStateManagerFile(*cell_state_cls=None*)
Bases: nova.cells.state.CellStateManager

cell_create(*ctxt*, *values*)

cell_delete(*ctxt*, *cell_name*)

cell_update(*ctxt*, *cell_name*, *values*)

sync_after(*f*)
Use as a decorator to wrap methods that update cell information in the database to make sure the data is synchronized immediately.

sync_before(*f*)
Use as a decorator to wrap methods that use cell information to make sure they sync the latest information from the DB periodically.

3.7.278 The nova.cells.utils Module

Cells Utility Methods

class ComputeNodeProxy(*obj*, *cell_path*)
Bases: nova.cells.utils._CellProxy

class ProxyObjectSerializer
Bases: nova.objects.base.NovaObjectSerializer

class ServiceProxy(*obj*, *cell_path*)
Bases: nova.cells.utils._CellProxy

add_cell_to_compute_node(*compute_node*, *cell_name*)
Fix compute_node attributes that should be unique. Allows API cell to query the 'id' by cell@id.

add_cell_to_service(*service*, *cell_name*)
Fix service attributes that should be unique. Allows API cell to query the 'id' or 'host' by cell@id/host.

add_cell_to_task_log(*task_log*, *cell_name*)
Fix task_log attributes that should be unique. In particular, the 'id' and 'host' fields should be prepended with cell name.

cell_with_item(*cell_name*, *item*)
Turn cell_name and item into <cell_name>@<item>.

get_instances_to_sync(*context*, *updated_since=None*, *project_id=None*, *deleted=True*, *shuffle=False*, *uuids_only=False*)
Return a generator that will return a list of active and deleted instances to sync with parent cells. The list may optionally be shuffled for periodic updates so that multiple cells services aren't self-healing the same instances in nearly lockstep.

split_cell_and_item(*cell_and_item*)
Split a combined cell@item and return them.

3.7.279 The nova.cells.weights.mute_child Module

If a child cell hasn't sent capacity or capability updates in a while, downgrade its likelihood of being chosen for scheduling requests.

class MuteChildWeigher
Bases: nova.cells.weights.BaseCellWeigher

If a child cell hasn't been heard from, greatly lower its selection weight.

MUTE_WEIGH_VALUE = 1.0

weight_multiplier()

3.7.280 The nova.cells.weights.ram_by_instance_type Module

Weigh cells by memory needed in a way that spreads instances.

class RamByInstanceTypeWeigher
Bases: nova.cells.weights.BaseCellWeigher

Weigh cells by instance_type requested.

weight_multiplier()

3.7.281 The nova.cells.weights.weight_offset Module

Weigh cells by their weight_offset in the DB. Cells with higher weight_offsets in the DB will be preferred.

class WeightOffsetWeigher
Bases: nova.cells.weights.BaseCellWeigher

Weight cell by weight_offset db field. Originally designed so you can set a default cell by putting its weight_offset to 999999999999999 (highest weight wins)

weight_multiplier()

3.7.282 The nova.cert.manager Module

Cert manager manages x509 certificates.

Related Flags

cert_topic What rpc topic to listen to (default: *cert*).

cert_manager The module name of a class derived from manager.Manager (default: nova.cert.manager.Manager).

class CertManager(**args*, ***kwargs*)
Bases: nova.manager.Manager

decrypt_text(*context*, *project_id*, *text*)
Decrypt base64 encoded text using the projects private key.

fetch_ca(*context*, *project_id*)
Get root ca for a project.

fetch_crl(*context*, *project_id*)
Get crl for a project.

generate_x509_cert(*context*, *user_id*, *project_id*)
Generate and sign a cert for user in project.

init_host()

revoke_certs_by_project(*context*, *project_id*)
Revoke all project certs.

revoke_certs_by_user(*context*, *user_id*)
Revoke all user certs.

revoke_certs_by_user_and_project(*context*, *user_id*, *project_id*)
Revoke certs for user in project.

target = <Target version=2.0>

3.7.283 The nova.cert.rpcapi Module

Client side of the cert manager RPC API.

class CertAPI
Bases: object

Client side of the cert rpc API.

API version history:

1.0 - Initial version. 1.1 - Added get_backdoor_port()

... Grizzly and Havana support message version 1.1. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.1.

2.0 - Major API rev for Icehouse

... Icehouse, Juno and Kilo support message version 2.0. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.0.

VERSION_ALIASES = {'kilo': '2.0', 'grizzly': '1.1', 'havana': '1.1', 'juno': '2.0', 'icehouse': '2.0'}

decrypt_text(*ctxt*, *project_id*, *text*)

fetch_ca(*ctxt*, *project_id*)

fetch_crl(*ctxt*, *project_id*)

generate_x509_cert(*ctxt*, *user_id*, *project_id*)

revoke_certs_by_project(*ctxt*, *project_id*)

revoke_certs_by_user(*ctxt*, *user_id*)

revoke_certs_by_user_and_project(*ctxt*, *user_id*, *project_id*)

3.7.284 The nova.cloudpipe.pipelib Module

CloudPipe - Build a user-data payload zip file, and launch an instance with it.

class CloudPipe(*skip_policy_check=False*)

Bases: object

get_encoded_zip(*project_id*)

launch_vpn_instance(*context*)

setup_key_pair(*context*)

setup_security_group(*context*)

is_vpn_image(*image_id*)

3.7.285 The nova.cmd.all Module

Starter script for all nova services.

This script attempts to start all the nova services in one process. Each service is started in its own greenthread. Please note that exceptions and sys.exit() on the starting of a service are logged and the script will continue attempting to launch the rest of the services.

main()

3.7.286 The nova.cmd.api Module

Starter script for Nova API.

Starts both the EC2 and OpenStack APIs in separate greenthreads.

main()

3.7.287 The nova.cmd.api_ec2 Module

Starter script for Nova EC2 API.

main()

3.7.288 The nova.cmd.api_metadata Module

Starter script for Nova Metadata API.

main()

3.7.289 The nova.cmd.api_os_compute Module

Starter script for Nova OS API.

main()

3.7.290 The nova.cmd.baseproxy Module

Base proxy module used to create compatible consoles for Openstack Nova.

exit_with_error(*msg*, *errno=-1*)

proxy(*host*, *port*)

3.7.291 The nova.cmd.cells Module

Starter script for Nova Cells Service.

main()

3.7.292 The nova.cmd.cert Module

Starter script for Nova Cert.

main()

3.7.293 The nova.cmd.compute Module

Starter script for Nova Compute.

block_db_access()

main()

3.7.294 The nova.cmd.conductor Module

Starter script for Nova Conductor.

main()

3.7.295 The nova.cmd.console Module

Starter script for Nova Console Proxy.

main()

3.7.296 The nova.cmd.consoleauth Module

VNC Console Proxy Server.

main()

3.7.297 The nova.cmd.dhcpbridge Module

Handle lease database updates from DHCP servers.

add_action_parsers(*subparsers*)

add_lease(*mac*, *ip_address*)
Set the IP that was assigned by the DHCP server.

block_db_access()

del_lease(*mac*, *ip_address*)
Called when a lease expires.

init_leases(*network_id*)
Get the list of hosts for a network.

main()
Parse environment and arguments and call the appropriate action.

old_lease(*mac*, *ip_address*)
Called when an old lease is recognized.

3.7.298 The nova.cmd.idmapshift Module

IDMapShift

IDMapShift is a tool that properly sets the ownership of a filesystem for use with linux user namespaces.

Usage

nova-idmapshift -i -u 0:10000:2000 -g 0:10000:2000 path

This command will idempotently shift *path* to proper ownership using the provided uid and gid mappings.

Arguments

nova-idmapshift -i -c -d -v -u [[guest-uid:host-uid:count],...] -g [[guest-gid:host-gid:count],...] -n [nobody-id] path

path: Root path of the filesystem to be shifted

-i, –idempotent: Shift operation will only be performed if filesystem appears unshifted

-c, –confirm: Will perform check on filesystem Returns 0 when filesystem appears shifted Returns 1 when filesystem appears unshifted

-d, –dry-run: Print chown operations, but won't perform them

-v, –verbose: Print chown operations while performing them

-u, –uid: User ID mappings, maximum of 3 ranges

-g, –gid: Group ID mappings, maximum of 3 ranges

-n, –nobody: ID to map all unmapped uid and gids to.

Purpose

When using user namespaces with linux containers, the filesystem of the container must be owned by the targeted user and group ids being applied to that container. Otherwise, processes inside the container won't be able to access the filesystem.

For example, when using the id map string '0:10000:2000', this means that user ids inside the container between 0 and 1999 will map to user ids on the host between 10000 and 11999. Root (0) becomes 10000, user 1 becomes 10001, user 50 becomes 10050 and user 1999 becomes 11999. This means that files that are owned by root need to actually be owned by user 10000, and files owned by 50 need to be owned by 10050, and so on.

IDMapShift will take the uid and gid strings used for user namespaces and properly set up the filesystem for use by those users. Uids and gids outside of provided ranges will be mapped to nobody (max uid/gid) so that they are inaccessible inside the container.

confirm_dir(*fsdir*, *uid_mappings*, *gid_mappings*, *nobody*)

confirm_path(*path*, *uid_ranges*, *gid_ranges*, *nobody*)

find_target_id(*fsid*, *mappings*, *nobody*, *memo*)

get_ranges(*maps*)

id_map_type(*val*)

main()

print_chown(*path*, *uid*, *gid*, *target_uid*, *target_gid*)

shift_dir(*fsdir*, *uid_mappings*, *gid_mappings*, *nobody*, *dry_run=False*, *verbose=False*)

shift_path(*path*, *uid_mappings*, *gid_mappings*, *nobody*, *uid_memo*, *gid_memo*, *dry_run=False*, *verbose=False*)

3.7.299 The nova.cmd.manage Module

CLI interface for nova management.

AccountCommands
alias of ProjectCommands

class **AgentBuildCommands**
Bases: object

Class for managing agent builds.

create(*os*, *architecture*, *version*, *url*, *md5hash*, *hypervisor='xen'*)
Creates a new agent build.

delete(*os*, *architecture*, *hypervisor='xen'*)
Deletes an existing agent build.

list(*hypervisor=None*)
Lists all agent builds.

arguments: <none>

modify(*os*, *architecture*, *version*, *url*, *md5hash*, *hypervisor='xen'*)
Update an existing agent build.

class ApiDbCommands
Bases: object

Class for managing the api database.

sync(*version=None*)
Sync the database up to the most recent version.

version()
Print the current database version.

class CellCommands
Bases: object

Commands for managing cells.

create(*name*, *cell_type='child'*, *username=None*, *broker_hosts=None*, *password=None*, *hostname=None*, *port=None*, *virtual_host=None*, *woffset=None*, *wscale=None*)

delete(*cell_name*)

list()

class DbCommands
Bases: object

Class for managing the main database.

archive_deleted_rows(*max_rows*)
Move up to max_rows deleted rows from production tables to shadow tables.

contract(*dry_run*, *force_experimental_contract=False*)
Contract database schema.

expand(*dry_run*)
Expand database schema.

migrate(*dry_run*)
Migrate database schema.

null_instance_uuid_scan(*delete=False*)
Lists and optionally deletes database records where instance_uuid is NULL.

sync(*version=None*)
Sync the database up to the most recent version.

version()
Print the current database version.

class FixedIpCommands
Bases: object

Class for managing fixed ip.

list(*host=None*)
Lists all fixed ips (optionally by host).

reserve(*address*)
Mark fixed ip as reserved

arguments: address

unreserve (*address*)
Mark fixed ip as free to use

arguments: address

class FloatingIpCommands
Bases: object

Class for managing floating ip.

static address_to_hosts (*addresses*)
Iterate over hosts within an address range.

If an explicit range specifier is missing, the parameter is interpreted as a specific individual address.

create (*ip_range*, *pool=None*, *interface=None*)
Creates floating ips for zone by range.

delete (*ip_range*)
Deletes floating ips by range.

list (*host=None*)
Lists all floating ips (optionally by host).

Note: if host is given, only active floating IPs are returned

class GetLogCommands
Bases: object

Get logging information.

errors ()
Get all of the errors from the log files.

syslog (*num_entries=10*)
Get <num_entries> of the nova syslog events.

class HostCommands
Bases: object

List hosts.

list (*zone=None*)
Show a list of all physical hosts. Filter by zone. args: [zone]

class NetworkCommands
Bases: object

Class for managing networks.

create (*label=None*, *cidr=None*, *num_networks=None*, *network_size=None*, *multi_host=None*, *vlan=None*, *vlan_start=None*, *vpn_start=None*, *cidr_v6=None*, *gateway=None*, *gateway_v6=None*, *bridge=None*, *bridge_interface=None*, *dns1=None*, *dns2=None*, *project_id=None*, *priority=None*, *uuid=None*, *fixed_cidr=None*)
Creates fixed ips for host by range.

delete (*fixed_range=None*, *uuid=None*)
Deletes a network.

list ()
List all created networks.

modify (*fixed_range*, *project=None*, *host=None*, *dis_project=None*, *dis_host=None*)
Associate/Disassociate Network with Project and/or Host arguments: network project host leave any field blank to ignore it

class ProjectCommands
Bases: object

Class for managing projects.

quota(*project_id*, *user_id=None*, *key=None*, *value=None*)
Create, update or display quotas for project/user

If no quota key is provided, the quota will be displayed. If a valid quota key is provided and it does not exist, it will be created. Otherwise, it will be updated.

scrub(*project_id*)
Deletes data associated with project.

class ServiceCommands
Bases: object

Enable and disable running services.

describe_resource(*host*)
Describes cpu/memory/hdd info for host.

Parameters **host** – hostname.

disable(*host*, *service*)
Disable scheduling for a service.

enable(*host*, *service*)
Enable scheduling for a service.

list(*host=None*, *service=None*)
Show a list of all running services. Filter by host & service name

class ShellCommands
Bases: object

bpython()
Runs a bpython shell.

Falls back to Ipython/python shell if unavailable

ipython()
Runs an Ipython shell.

Falls back to Python shell if unavailable

python()
Runs a python shell.

Falls back to Python shell if unavailable

run(*shell=None*)
Runs a Python interactive interpreter.

script(*path*)
Runs the script from the specified path with flags set properly.

arguments: path

class VmCommands
Bases: object

Class for mangaging VM instances.

list(*host=None*)
Show a list of all instances.

class VpnCommands
Bases: object

Class for managing VPNs.

change(*project_id*, *ip*, *port*)
Change the ip and port for a vpn.

this will update all networks associated with a project not sure if that's the desired behavior or not, patches accepted

add_command_parsers(*subparsers*)

args(**args*, ***kwargs*)

main()
Parse options and call the appropriate class/method.

methods_of(*obj*)
Get all callable methods of an object that don't start with underscore

returns a list of tuples of the form (method_name, method)

param2id(*object_id*)
Helper function to convert various volume id types to internal id. args: [object_id], e.g. 'vol-0000000a' or 'volume-0000000a' or '10'

validate_network_plugin(*f*)
Decorator to validate the network plugin.

3.7.300 The nova.cmd.network Module

Starter script for Nova Network.

block_db_access()

main()

3.7.301 The nova.cmd.novnc Module

3.7.302 The nova.cmd.novncproxy Module

Websocket proxy that is compatible with OpenStack Nova noVNC consoles. Leverages websockify.py by Joel Martin

main()

3.7.303 The nova.cmd.objectstore Module

Daemon for nova objectstore. Supports S3 API.

main()

3.7.304 The nova.cmd.scheduler Module

Starter script for Nova Scheduler.

main()

3.7.305 The nova.cmd.serialproxy Module

Websocket proxy that is compatible with OpenStack Nova Serial consoles. Leverages websockify.py by Joel Martin. Based on nova-novncproxy.

main()

3.7.306 The nova.cmd.spicehtml5proxy Module

Websocket proxy that is compatible with OpenStack Nova SPICE HTML5 consoles. Leverages websockify.py by Joel Martin

main()

3.7.307 The nova.cmd.xvpvncproxy Module

XVP VNC Console Proxy Server.

main()

3.7.308 The nova.compute.api Module

Handles all requests relating to compute resources (e.g. guest VMs, networking and storage of VMs, and compute hosts on which they run).

class API(*image_api=None*, *network_api=None*, *volume_api=None*, *security_group_api=None*, *skip_policy_check=False*, ***kwargs*)

Bases: nova.db.base.Base

API for interacting with the compute manager.

add_fixed_ip(*context*, *target*, **args*, ***kwargs*)

Add fixed_ip from specified network to given instance.

attach_interface(*context*, *target*, **args*, ***kwargs*)

Use hotplug to add an network adapter to an instance.

attach_volume(*context*, *target*, **args*, ***kwargs*)

Attach an existing volume to an existing instance.

backup(*context*, *target*, **args*, ***kwargs*)

cell_type

compute_task_api

confirm_resize(*context*, *target*, **args*, ***kwargs*)

create(**args*, ***kwargs*)

Provision instances, sending instance information to the scheduler. The scheduler will determine where the instance(s) go and will handle creating the DB entries.

Returns a tuple of (instances, reservation_id)

create_db_entry_for_new_instance(*context*, *instance_type*, *image*, *instance*, *security_group*, *block_device_mapping*, *num_instances*, *index*, *shutdown_terminate=False*)

Create an entry in the DB for this new instance, including any related table updates (such as security group, etc).

This is called by the scheduler after a location for the instance has been determined.

delete(*context*, *target*, **args*, ***kwargs*)

delete_instance_metadata(*context*, *target*, **args*, ***kwargs*)
Delete the given metadata item from an instance.

detach_interface(*context*, *target*, **args*, ***kwargs*)
Detach an network adapter from an instance.

detach_volume(*context*, *target*, **args*, ***kwargs*)
Detach a volume from an instance.

evacuate(*context*, *instance*, **args*, ***kw*)
Running evacuate to target host.

Checking vm compute host state, if the host not in expected_state, raising an exception.

Parameters

- **instance** – The instance to evacuate
- **host** – Target host. if not set, the scheduler will pick up one
- **on_shared_storage** – True if instance files on shared storage
- **admin_password** – password to set on rebuilt instance

external_instance_event(*context*, *instances*, *events*)

force_delete(*context*, *target*, **args*, ***kwargs*)
Force delete an instance in any vm_state/task_state.

force_stop(*context*, *instance*, *do_cast=True*, *clean_shutdown=True*)

get(*context*, *instance_id*, *want_objects=False*, *expected_attrs=None*)
Get a single instance with the given instance_id.

get_all(*context*, *search_opts=None*, *limit=None*, *marker=None*, *want_objects=False*, *expected_attrs=None*, *sort_keys=None*, *sort_dirs=None*)
Get all instances filtered by one of the given parameters.

If there is no filter and the context is an admin, it will retrieve all instances in the system.

Deleted instances will be returned by default, unless there is a search option that says otherwise.

The results will be sorted based on the list of sort keys in the 'sort_keys' parameter (first value is primary sort key, second value is secondary sort ket, etc.). For each sort key, the associated sort direction is based on the list of sort directions in the 'sort_dirs' parameter.

get_all_instance_metadata(*context*, *search_filts*)

get_all_system_metadata(*context*, *search_filts*)

get_console_output(*context*, *target*, **args*, ***kwargs*)
Get console output for an instance.

get_diagnostics(*context*, *target*, **args*, ***kwargs*)
Retrieve diagnostics for the given instance.

get_instance_diagnostics(*context*, *target*, **args*, ***kwargs*)
Retrieve diagnostics for the given instance.

get_instance_faults(*context*, *instances*)
Get all faults for a list of instance uuids.

get_instance_metadata(*context*, *target*, **args*, ***kwargs*)
Get all metadata associated with an instance.

get_lock(*context*, *target*, **args*, ***kwargs*)
Return the boolean state of given instance's lock.

get_migrations(*context*, *filters*)
Get all migrations for the given filters.

get_rdp_connect_info(*context*, *instance*, **args*, ***kwargs*)
Used in a child cell to get console info.

get_rdp_console(*context*, *target*, **args*, ***kwargs*)
Get a url to an instance Console.

get_serial_console(*context*, *target*, **args*, ***kwargs*)
Get a url to a serial console.

get_serial_console_connect_info(*context*, *instance*, **args*, ***kwargs*)
Used in a child cell to get serial console.

get_spice_connect_info(*context*, *instance*, **args*, ***kwargs*)
Used in a child cell to get console info.

get_spice_console(*context*, *target*, **args*, ***kwargs*)
Get a url to an instance Console.

get_vnc_connect_info(*context*, *instance*, **args*, ***kwargs*)
Used in a child cell to get console info.

get_vnc_console(*context*, *target*, **args*, ***kwargs*)
Get a url to an instance Console.

inject_network_info(*context*, *target*, **args*, ***kwargs*)

is_expected_locked_by(*context*, *instance*)

is_volume_backed_instance(*context*, *instance*, *bdms=None*)

live_migrate(*context*, *instance*, **args*, ***kwargs*)

lock(*context*, *target*, **args*, ***kwargs*)
Lock the given instance.

pause(*context*, *target*, **args*, ***kwargs*)

reboot(*context*, *target*, **args*, ***kwargs*)
Reboot the given instance.

rebuild(*context*, *target*, **args*, ***kwargs*)

remove_fixed_ip(*context*, *target*, **args*, ***kwargs*)
Remove fixed_ip from specified network to given instance.

rescue(*context*, *target*, **args*, ***kwargs*)
Rescue the given instance.

reset_network(*context*, *target*, **args*, ***kwargs*)

resize(*context*, *target*, **args*, ***kwargs*)

restore(*context*, *target*, **args*, ***kwargs*)
Restore a previously deleted (but not reclaimed) instance.

resume(*context*, *target*, **args*, ***kwargs*)

revert_resize(*context*, *target*, **args*, ***kwargs*)

set_admin_password(*context*, *target*, **args*, ***kwargs*)

shelve(*context*, *target*, **args*, ***kwargs*)
Shelve an instance.

Shuts down an instance and frees it up to be removed from the hypervisor.

shelve_offload(*context*, *target*, **args*, ***kwargs*)
Remove a shelved instance from the hypervisor.

snapshot(*context*, *target*, **args*, ***kwargs*)

snapshot_volume_backed(*context*, *instance*, **args*, ***kw*)
Snapshot the given volume-backed instance.

Parameters

- **instance** – nova.objects.instance.Instance object
- **image_meta** – metadata for the new image
- **name** – name of the backup or snapshot
- **extra_properties** – dict of extra image properties to include

Returns the new image metadata

soft_delete(*context*, *target*, **args*, ***kwargs*)

start(*context*, *instance*, **args*, ***kwargs*)

stop(*context*, *instance*, **args*, ***kwargs*)

suspend(*context*, *target*, **args*, ***kwargs*)

swap_volume(*context*, *target*, **args*, ***kwargs*)
Swap volume attached to an instance.

unlock(*context*, *target*, **args*, ***kwargs*)
Unlock the given instance.

unpause(*context*, *target*, **args*, ***kwargs*)

unrescue(*context*, *target*, **args*, ***kwargs*)
Unrescue the given instance.

unshelve(*context*, *target*, **args*, ***kwargs*)
Restore a shelved instance.

update_instance_metadata(*context*, *target*, **args*, ***kwargs*)
Updates or creates instance metadata.

If delete is True, metadata items that are not specified in the *metadata* argument will be deleted.

volume_snapshot_create(*context*, *target*, **args*, ***kwargs*)

volume_snapshot_delete(*context*, *target*, **args*, ***kwargs*)

class AggregateAPI(***kwargs*)
Bases: nova.db.base.Base

Sub-set of the Compute Manager API for managing host aggregates.

add_host_to_aggregate(*context*, **args*, ***kw*)
Adds the host to an aggregate.

create_aggregate(*context*, **args*, ***kw*)
Creates the model for the aggregate.

delete_aggregate(*context*, **args*, ***kw*)
Deletes the aggregate.

get_aggregate(*context*, *aggregate_id*)
Get an aggregate by id.

get_aggregate_list(*context*)
Get all the aggregates.

is_safe_to_update_az(*context*, *metadata*, *aggregate*, *hosts=None*, *action_name='Add'*)
Determine if updates alter an aggregate's availability zone.

Parameters

- **context** – local context
- **metadata** – Target metadata for updating aggregate
- **aggregate** – Aggregate to update
- **hosts** (*list*) – Hosts to check. If None, aggregate.hosts is used

Action_name Calling method for logging purposes

remove_host_from_aggregate(*context*, **args*, ***kw*)
Removes host from the aggregate.

update_aggregate(*context*, **args*, ***kw*)
Update the properties of an aggregate.

update_aggregate_metadata(*context*, **args*, ***kw*)
Updates the aggregate metadata.

class HostAPI(*rpcapi=None*)
Bases: nova.db.base.Base

Sub-set of the Compute Manager API for managing host operations.

compute_node_get(*context*, *compute_id*)
Return compute node entry for particular integer ID.

compute_node_get_all(*context*)

compute_node_search_by_hypervisor(*context*, *hypervisor_match*)

compute_node_statistics(*context*)

get_host_uptime(*context*, *host_name*)
Returns the result of calling "uptime" on the target host.

host_power_action(*context*, **args*, ***kw*)
Reboots, shuts down or powers up the host.

instance_get_all_by_host(*context*, *host_name*)
Return all instances on the given host.

service_delete(*context*, *service_id*)
Deletes the specified service.

service_get_all(*context*, *filters=None*, *set_zones=False*)
Returns a list of services, optionally filtering the results.

If specified, 'filters' should be a dictionary containing services attributes and matching values. Ie, to get a list of services for the 'compute' topic, use filters={'topic': 'compute'}.

service_get_by_compute_host(*context*, *host_name*)
Get service entry for the given compute hostname.

service_update(*context*, *host_name*, *binary*, *params_to_update*)
Enable / Disable a service.

For compute services, this stops new builds and migrations going to the host.

set_host_enabled(*context*, **args*, ***kw*)
Sets the specified host's ability to accept new instances.

set_host_maintenance(*context*, **args*, ***kw*)
Start/Stop host maintenance window. On start, it triggers guest VMs evacuation.

task_log_get_all(*context*, *task_name*, *period_beginning*, *period_ending*, *host=None*, *state=None*)
Return the task logs within a given range, optionally filtering by host and/or state.

class InstanceActionAPI(*db_driver=None*)
Bases: nova.db.base.Base

Sub-set of the Compute Manager API for managing instance actions.

action_events_get(*context*, *instance*, *action_id*)

action_get_by_request_id(*context*, *instance*, *request_id*)

actions_get(*context*, *instance*)

class KeypairAPI(*db_driver=None*)
Bases: nova.db.base.Base

Subset of the Compute Manager API for managing key pairs.

create_key_pair(*context*, **args*, ***kw*)
Create a new key pair.

delete_key_pair(*context*, **args*, ***kw*)
Delete a keypair by name.

get_key_pair(*context*, *user_id*, *key_name*)
Get a keypair by name.

get_key_pairs(*context*, *user_id*)
List key pairs.

get_notifier = <functools.partial object at 0x7f858b9c2f70>

import_key_pair(*context*, **args*, ***kw*)
Import a key pair using an existing public key.

wrap_exception = <functools.partial object at 0x7f858b9c2fc8>

class SecurityGroupAPI(*skip_policy_check=False*, ***kwargs*)
Bases: nova.db.base.Base, nova.network.security_group.security_group_base.SecurityGroupBas

Sub-set of the Compute API related to managing security groups and security group rules

add_default_rules(*context*, *vals*)

add_rules(*context*, *id*, *name*, *vals*)
Add security group rule(s) to security group.

Note: the Nova security group API doesn't support adding multiple security group rules at once but the EC2 one does. Therefore, this function is written to support both.

add_to_instance(*context*, *target*, **args*, ***kwargs*)
Add security group to the instance.

create_security_group(*context*, *name*, *description*)

default_rule_exists(*context*, *values*)
Indicates whether the specified rule values are already defined in the default security group rules.

destroy(*context*, *security_group*)

ensure_default(*context*)
Ensure that a context has a security group.

Creates a security group for the security context if it does not already exist.

Parameters **context** – the security context

get(*context*, *name=None*, *id=None*, *map_exception=False*)

get_all_default_rules(*context*)

get_default_rule(*context*, *id*)

get_instance_security_groups(*context*, *instance_uuid*, *detailed=False*)

get_rule(*context*, *id*)

id_is_uuid = False

is_associated_with_server(*security_group*, *instance_uuid*)
Check if the security group is already associated with the instance. If Yes, return True.

list(*context*, *names=None*, *ids=None*, *project=None*, *search_opts=None*)

populate_security_groups(*instance*, *security_groups*)

remove_default_rules(*context*, *rule_ids*)

remove_from_instance(*context*, *target*, **args*, ***kwargs*)
Remove the security group associated with the instance.

remove_rules(*context*, *security_group*, *rule_ids*)

trigger_members_refresh(*context*, *group_ids*)
Called when a security group gains a new or loses a member.

Sends an update request to each compute node for each instance for which this is relevant.

trigger_rules_refresh(*context*, *id*)
Called when a rule is added to or removed from a security_group.

update_security_group(*context*, *security_group*, *name*, *description*)

validate_id(*id*)

validate_property(*value*, *property*, *allowed*)
Validate given security group property.

Parameters

- **value** – the value to validate, as a string or unicode
- **property** – the property, either 'name' or 'description'
- **allowed** – the range of characters allowed

check_instance_cell(*fn*)

check_instance_host(*function*)

check_instance_lock(*function*)

check_instance_state(*vm_state=None*, *task_state=(None,)*, *must_have_launched=True*)
Decorator to check VM and/or task state before entry to API functions.

If the instance is in the wrong state, or has not been successfully started at least once the wrapper will raise an exception.

check_policy(*context*, *action*, *target*, *scope='compute'*)

policy_decorator(*scope*)
Check corresponding policy prior of wrapped method to execution.

wrap_check_policy(*func*)

wrap_check_security_groups_policy(*func*)

3.7.309 The nova.compute.arch Module

Constants and helper APIs for dealing with CPU architectures

The constants provide the standard names for all known processor architectures. Many have multiple variants to deal with big-endian vs little-endian modes, as well as 32 vs 64 bit word sizes. These names are chosen to be identical to the architecture names expected by libvirt, so if ever adding new ones, ensure it matches libvirt's expectation.

canonicalize(*name*)
Canonicalize the architecture name

Parameters **name** – architecture name to canonicalize

Returns a canonical architecture name

from_host()
Get the architecture of the host OS

Returns the canonicalized host architecture

is_valid(*name*)
Check if a string is a valid architecture

Parameters **name** – architecture name to validate

Returns True if @name is valid

3.7.310 The nova.compute.build_results Module

Possible results from instance build

Results represent the ultimate result of an attempt to build an instance.

Results describe whether an instance was actually built, failed to build, or was rescheduled.

3.7.311 The nova.compute.cells_api Module

Compute API that proxies via Cells Service.

class ComputeCellsAPI(**args*, ***kwargs*)
Bases: nova.compute.api.API

add_fixed_ip(*context*, *instance*, **args*, ***kwargs*)

associate_floating_ip(*context*, *target*, **args*, ***kwargs*)

create(**args*, ***kwargs*)
We can use the base functionality, but I left this here just for completeness.

delete(*context*, *instance*)

delete_instance_metadata(*context*, *instance*, **args*, ***kwargs*)

evacuate(*context*, *instance*, **args*, ***kwargs*)

force_delete(*context*, *instance*, **args*, ***kwargs*)

get_console_output(*context*, *instance*, **args*, ***kwargs*)

get_diagnostics(*context*, *instance*)
Retrieve diagnostics for the given instance.

get_instance_diagnostics(*context*, *instance*)
Retrieve diagnostics for the given instance.

get_migrations(*context*, *filters*)

get_rdp_console(*context*, *target*, **args*, ***kwargs*)

get_serial_console(*context*, *target*, **args*, ***kwargs*)

get_spice_console(*context*, *target*, **args*, ***kwargs*)

get_vnc_console(*context*, *target*, **args*, ***kwargs*)

remove_fixed_ip(*context*, *instance*, **args*, ***kwargs*)

rescue(*context*, *instance*, **args*, ***kwargs*)

restore(*context*, *instance*, **args*, ***kwargs*)

shelve(*context*, *target*, **args*, ***kwargs*)

shelve_offload(*context*, *target*, **args*, ***kwargs*)

soft_delete(*context*, *instance*)

unrescue(*context*, *instance*, **args*, ***kwargs*)

unshelve(*context*, *target*, **args*, ***kwargs*)

update_instance_metadata(*context*, *target*, **args*, ***kwargs*)

class ComputeRPCAPIRedirect(*cells_rpcapi*)
Bases: object

cells_compatible = ['start_instance', 'stop_instance', 'reboot_instance', 'suspend_instance', 'resume_instance', 'terr

class ComputeRPCProxyAPI
Bases: nova.compute.rpcapi.ComputeAPI

Class used to substitute Compute RPC API that will proxy via the cells manager to a compute manager in a child cell.

get_client(*target*, *version_cap*, *serializer*)

class ConductorTaskRPCAPIRedirect(*cells_rpcapi_obj*)
Bases: object

cells_compatible = ['build_instances', 'resize_instance', 'live_migrate_instance', 'rebuild_instance']

class HostAPI

Bases: nova.compute.api.HostAPI

HostAPI() class for cells.

Implements host management related operations. Works by setting the RPC API used by the base class to proxy via the cells manager to the compute manager in the correct cell. Hosts specified with cells will need to be of the format 'path!to!cell@host'.

DB methods in the base class are also overridden to proxy via the cells manager.

compute_node_get(*context*, *compute_id*)

Get a compute node from a particular cell by its integer ID. compute_id should be in the format of 'path!to!cell@ID'.

compute_node_get_all(*context*)

compute_node_search_by_hypervisor(*context*, *hypervisor_match*)

compute_node_statistics(*context*)

get_host_uptime(*context*, *host_name*)

Returns the result of calling "uptime" on the target host.

host_power_action(*context*, *host_name*, *action*)

instance_get_all_by_host(*context*, *host_name*)

Get all instances by host. Host might have a cell prepended to it, so we'll need to strip it out. We don't need to proxy this call to cells, as we have instance information here in the API cell.

service_delete(*context*, *service_id*)

Deletes the specified service.

service_get_all(*context*, *filters=None*, *set_zones=False*)

service_get_by_compute_host(*context*, *host_name*)

service_update(*context*, *host_name*, *binary*, *params_to_update*)

Used to enable/disable a service. For compute services, setting to disabled stops new builds arriving on that host.

Parameters

- **host_name** – the name of the host machine that the service is running
- **binary** – The name of the executable that the service runs as
- **params_to_update** – eg. {'disabled': True}

set_host_enabled(*context*, *host_name*, *enabled*)

task_log_get_all(*context*, *task_name*, *beginning*, *ending*, *host=None*, *state=None*)

Return the task logs within a given range from cells, optionally filtering by the host and/or state. For cells, the host should be a path like 'path!to!cell@host'. If no @host is given, only task logs from a particular cell will be returned.

class InstanceActionAPI

Bases: nova.compute.api.InstanceActionAPI

InstanceActionAPI() class for cells.

action_events_get(*context*, *instance*, *action_id*)

action_get_by_request_id(*context*, *instance*, *request_id*)

actions_get(*context*, *instance*)

class RPCClientCellsProxy(*target*, *version_cap*)
Bases: object

call(*ctxt*, *method*, ***kwargs*)

can_send_version(*version*)

cast(*ctxt*, *method*, ***kwargs*)

prepare(***kwargs*)

3.7.312 The nova.compute.claims Module

Claim objects for use with resource tracking.

class Claim(*context*, *instance*, *tracker*, *resources*, *overhead=None*, *limits=None*)
Bases: nova.compute.claims.NopClaim

A declaration that a compute host operation will require free resources. Claims serve as marker objects that resources are being held until the update_available_resource audit process runs to do a full reconciliation of resource usage.

This information will be used to help keep the local compute hosts's ComputeNode model in sync to aid the scheduler in making efficient / more correct decisions with respect to host selection.

abort()
Compute operation requiring claimed resources has failed or been aborted.

disk_gb

memory_mb

numa_topology

class MoveClaim(*context*, *instance*, *instance_type*, *image_meta*, *tracker*, *resources*, *overhead=None*, *limits=None*)
Bases: nova.compute.claims.Claim

Claim used for holding resources for an incoming move operation.

Move can be either a migrate/resize, live-migrate or an evacuate operation.

abort()
Compute operation requiring claimed resources has failed or been aborted.

disk_gb

memory_mb

numa_topology

class NopClaim(*migration=None*)
Bases: object

For use with compute drivers that do not support resource tracking.

abort()

disk_gb

memory_mb

3.7.313 The nova.compute.cpumodel Module

3.7.314 The nova.compute.flavors Module

Built-in instance properties.

create(*name*, *memory*, *vcpus*, *root_gb*, *ephemeral_gb=0*, *flavorid=None*, *swap=0*, *rxtx_factor=1.0*, *is_public=True*)
Creates flavors.

delete_flavor_info(*metadata*, **prefixes*)
Delete flavor instance_type information from instance's system_metadata by prefix.

destroy(*name*)
Marks flavor as deleted.

extract_flavor(*instance*, *prefix=''*)
Create a Flavor object from instance's system_metadata information.

get_all_flavors(*ctxt=None*, *inactive=False*, *filters=None*)
Get all non-deleted flavors as a dict.

Pass inactive=True if you want deleted flavors returned also.

get_all_flavors_sorted_list(*ctxt=None*, *filters=None*, *sort_key='flavorid'*, *sort_dir='asc'*, *limit=None*, *marker=None*)
Get all non-deleted flavors as a sorted list.

get_default_flavor()
Get the default flavor.

get_flavor(*instance_type_id*, *ctxt=None*, *inactive=False*)
Retrieves single flavor by id.

get_flavor_access_by_flavor_id(*flavorid*, *ctxt=None*)
Retrieve flavor access list by flavor id.

get_flavor_by_flavor_id(*flavorid*, *ctxt=None*, *read_deleted='yes'*)
Retrieve flavor by flavorid.

Raises FlavorNotFound

get_flavor_by_name(*name*, *ctxt=None*)
Retrieves single flavor by name.

save_flavor_info(*metadata*, *instance_type*, *prefix=''*)
Save properties from instance_type into instance's system_metadata, in the format of:

[prefix]instance_type_[key]

This can be used to update system_metadata in place from a type, as well as stash information about another instance_type for later use (such as during resize).

validate_extra_spec_keys(*key_names_list*)

3.7.315 The nova.compute.hv_type Module

Constants and helper APIs for dealing with virtualization types

The constants provide the standard names for all known guest virtualization types. This is not to be confused with the Nova hypervisor driver types, since one driver may support multiple virtualization types and one virtualization type (eg 'xen') may be supported by multiple drivers ('XenAPI' or 'Libvirt-Xen').

canonicalize(*name*)
Canonicalize the hypervisor type name

Parameters **name** – hypervisor type name to canonicalize

Returns a canonical hypervisor type name

is_valid(*name*)
Check if a string is a valid hypervisor type

Parameters **name** – hypervisor type name to validate

Returns True if @name is valid

3.7.316 The nova.compute.instance_actions Module

Possible actions on an instance.

Actions should probably match a user intention at the API level. Because they can be user visible that should help to avoid confusion. For that reason they tend to maintain the casing sent to the API.

Maintaining a list of actions here should protect against inconsistencies when they are used.

3.7.317 The nova.compute.manager Module

Handles all processes relating to instances (guest vms).

The ComputeManager class is a nova.manager.Manager that handles RPC calls relating to creating instances. It is responsible for building a disk image, launching it via the underlying virtualization driver, responding to calls to check its state, attaching persistent storage, and terminating it.

class ComputeManager(*compute_driver=None*, **args*, ***kwargs*)
Bases: nova.manager.Manager

Manages the running instances from creation to destruction.

SHUTDOWN_RETRY_INTERVAL = 10

add_aggregate_host(*context*, **args*, ***kw*)
Notify hypervisor of change (for hypervisor pools).

add_fixed_ip_to_instance(*context*, **args*, ***kw*)
Calls network_api to add new fixed_ip to instance then injects the new network info and resets instance networking.

attach_interface(*context*, **args*, ***kw*)
Use hotplug to add an network adapter to an instance.

attach_volume(*context*, **args*, ***kw*)
Attach a volume to an instance.

backup_instance(*context*, **args*, ***kw*)
Backup an instance on this host.

Parameters

- **backup_type** – daily | weekly
- **rotation** – int representing how many backups to keep around

build_and_run_instance(*context*, **args*, ***kw*)

change_instance_metadata(*context*, **args*, ***kw*)
Update the metadata published to the instance.

check_can_live_migrate_destination(*context*, **args*, ***kw*)
Check if it is possible to execute live migration.

This runs checks on the destination host, and then calls back to the source host to check the results.

Parameters

- **context** – security context
- **instance** – dict of instance data
- **block_migration** – if true, prepare for block migration
- **disk_over_commit** – if true, allow disk over commit

Returns a dict containing migration info

check_can_live_migrate_source(*context*, **args*, ***kw*)
Check if it is possible to execute live migration.

This checks if the live migration can succeed, based on the results from check_can_live_migrate_destination.

Parameters

- **ctxt** – security context
- **instance** – dict of instance data
- **dest_check_data** – result of check_can_live_migrate_destination

Returns a dict containing migration info

check_instance_shared_storage(*context*, **args*, ***kw*)
Check if the instance files are shared

Parameters

- **ctxt** – security context
- **instance** – dict of instance data
- **data** – result of driver.check_instance_shared_storage_local

Returns True if instance disks located on shared storage and False otherwise.

cleanup_host()

confirm_resize(*context*, **args*, ***kw*)

detach_interface(*context*, **args*, ***kw*)
Detach an network adapter from an instance.

detach_volume(*context*, **args*, ***kw*)
Detach a volume from an instance.

external_instance_event(*context*, **args*, ***kw*)

finish_resize(*context*, **args*, ***kw*)
Completes the migration process.

Sets up the newly transferred disk and turns on the instance at its new host machine.

finish_revert_resize(*context*, **args*, ***kw*)
Finishes the second half of reverting a resize.

Bring the original source instance state back (active/shutoff) and revert the resized attributes in the database.

get_console_output(**args*, ***kwargs*)

get_console_pool_info(*context*, **args*, ***kw*)

get_console_topic(*context*)
Retrieves the console host for a project on this host.

Currently this is just set in the flags for each compute host.

get_diagnostics(*context*, **args*, ***kw*)
Retrieve diagnostics for an instance on this host.

get_host_uptime(*context*, **args*, ***kw*)
Returns the result of calling "uptime" on the target host.

get_instance_diagnostics(*context*, **args*, ***kwargs*)
Retrieve diagnostics for an instance on this host.

get_rdp_console(**args*, ***kwargs*)

get_serial_console(**args*, ***kwargs*)

get_spice_console(**args*, ***kwargs*)

get_vnc_console(**args*, ***kwargs*)

handle_events(*event*)

handle_lifecycle_event(*event*)

host_maintenance_mode(*context*, **args*, ***kw*)
Start/Stop host maintenance window. On start, it triggers guest VMs evacuation.

host_power_action(*context*, **args*, ***kw*)
Reboots, shuts down or powers up the host.

init_host()
Initialization for a standalone compute service.

init_virt_events()

inject_file(*context*, **args*, ***kw*)
Write a file to the specified path in an instance on this host.

inject_network_info(*context*, **args*, ***kwargs*)
Inject network info, but don't return the info.

live_migration(*context*, **args*, ***kw*)
Executing live migration.

Parameters

- **context** – security context
- **instance** – a nova.objects.instance.Instance object
- **dest** – destination host
- **block_migration** – if true, prepare for block migration
- **migration** – an nova.objects.Migration object

- **migrate_data** – implementation specific params

pause_instance(*context*, **args*, ***kw*)
Pause an instance on this host.

post_live_migration_at_destination(*context*, **args*, ***kw*)
Post operations for live migration .

Parameters

- **context** – security context
- **instance** – Instance dict
- **block_migration** – if true, prepare for block migration

pre_live_migration(*context*, **args*, ***kw*)
Preparations for live migration at dest host.

Parameters

- **context** – security context
- **instance** – dict of instance data
- **block_migration** – if true, prepare for block migration
- **migrate_data** – if not None, it is a dict which holds data required for live migration without shared storage.

pre_start_hook()
After the service is initialized, but before we fully bring the service up by listening on RPC queues, make sure to update our available resources (and indirectly our available nodes).

prep_resize(*context*, **args*, ***kw*)
Initiates the process of moving a running instance to another host.

Possibly changes the RAM and disk size in the process.

quiesce_instance(**args*, ***kwargs*)

reboot_instance(*context*, **args*, ***kw*)
Reboot an instance on this host.

rebuild_instance(**args*, ***kwargs*)

refresh_instance_security_rules(*context*, **args*, ***kwargs*)
Tell the virtualization driver to refresh security rules for an instance.

Passes straight through to the virtualization driver.

Synchronise the call because we may still be in the middle of creating the instance.

refresh_provider_fw_rules(*context*, **args*, ***kw*)
This call passes straight through to the virtualization driver.

refresh_security_group_members(*context*, **args*, ***kw*)
Tell the virtualization driver to refresh security group members.

Passes straight through to the virtualization driver.

refresh_security_group_rules(*context*, **args*, ***kw*)
Tell the virtualization driver to refresh security group rules.

Passes straight through to the virtualization driver.

remove_aggregate_host(*context*, **args*, ***kw*)
Removes a host from a physical hypervisor pool.

remove_fixed_ip_from_instance(*context*, **args*, ***kw*)
Calls network_api to remove existing fixed_ip from instance by injecting the altered network info and resetting instance networking.

remove_volume_connection(*context*, **args*, ***kw*)
Remove a volume connection using the volume api.

rescue_instance(*context*, **args*, ***kw*)

reserve_block_device_name(*context*, **args*, ***kw*)

reset_network(**args*, ***kwargs*)

resize_instance(*context*, **args*, ***kw*)
Starts the migration of a running instance to another host.

restore_instance(*context*, **args*, ***kw*)
Restore a soft-deleted instance on this host.

resume_instance(*context*, **args*, ***kw*)
Resume the given suspended instance.

revert_resize(*context*, **args*, ***kw*)
Destroys the new instance on the destination machine.

Reverts the model changes, and powers on the old instance on the source machine.

rollback_live_migration_at_destination(*context*, **args*, ***kw*)
Cleaning up image directory that is created pre_live_migration.

Parameters

- **context** – security context
- **instance** – a nova.objects.instance.Instance object sent over rpc

set_admin_password(*context*, **args*, ***kw*)
Set the root/admin password for an instance on this host.

This is generally only called by API password resets after an image has been built.

@param context: Nova auth context. @param instance: Nova instance object. @param new_pass: The admin password for the instance.

set_host_enabled(*context*, **args*, ***kw*)
Sets the specified host's ability to accept new instances.

shelve_instance(*context*, **args*, ***kw*)
Shelve an instance.

This should be used when you want to take a snapshot of the instance. It also adds system_metadata that can be used by a periodic task to offload the shelved instance after a period of time.

Parameters

- **context** – request context
- **instance** – an Instance object
- **image_id** – an image id to snapshot to.
- **clean_shutdown** – give the GuestOS a chance to stop

shelve_offload_instance(*context*, **args*, ***kw*)
Remove a shelved instance from the hypervisor.

This frees up those resources for use by other instances, but may lead to slower unshelve times for this instance. This method is used by volume backed instances since restoring them doesn't involve the potentially large download of an image.

Parameters

- **context** – request context
- **instance** – nova.objects.instance.Instance
- **clean_shutdown** – give the GuestOS a chance to stop

snapshot_instance(*context*, **args*, ***kw*)
Snapshot an instance on this host.

Parameters

- **context** – security context
- **instance** – a nova.objects.instance.Instance object
- **image_id** – glance.db.sqlalchemy.models.Image.Id

soft_delete_instance(*context*, **args*, ***kw*)
Soft delete an instance on this host.

start_instance(*context*, **args*, ***kw*)
Starting an instance on this host.

stop_instance(*context*, **args*, ***kw*)
Stopping an instance on this host.

suspend_instance(*context*, **args*, ***kw*)
Suspend the given instance.

swap_volume(*context*, **args*, ***kw*)
Swap volume for an instance.

target = <Target version=4.2>

terminate_instance(*context*, **args*, ***kw*)
Terminate an instance on this host.

unpause_instance(*context*, **args*, ***kw*)
Unpause a paused instance on this host.

unquiesce_instance(**args*, ***kwargs*)

unrescue_instance(*context*, **args*, ***kw*)

unshelve_instance(*context*, **args*, ***kw*)
Unshelve the instance.

Parameters

- **context** – request context
- **instance** – a nova.objects.instance.Instance object
- **image** – an image to build from. If None we assume a volume backed instance.
- **filter_properties** – dict containing limits, retry info etc.
- **node** – target compute node

update_available_resource(*context*)
See driver.get_available_resource()

Periodic process that keeps that the compute host's understanding of resource availability and usage in sync with the underlying hypervisor.

Parameters **context** – security context

validate_console_port(**args*, ***kwargs*)

volume_snapshot_create(**args*, ***kwargs*)

volume_snapshot_delete(**args*, ***kwargs*)

class ComputeVirtAPI(*compute*)
Bases: nova.virt.virtapi.VirtAPI

provider_fw_rule_get_all(*context*)

wait_for_instance_event(**args*, ***kwds*)
Plan to wait for some events, run some code, then wait.

This context manager will first create plans to wait for the provided event_names, yield, and then wait for all the scheduled events to complete.

Note that this uses an eventlet.timeout.Timeout to bound the operation, so callers should be prepared to catch that failure and handle that situation appropriately.

If the event is not received by the specified timeout deadline, eventlet.timeout.Timeout is raised.

If the event is received but did not have a 'completed' status, a NovaException is raised. If an error_callback is provided, instead of raising an exception as detailed above for the failure case, the callback will be called with the event_name and instance, and can return True to continue waiting for the rest of the events, False to stop processing, or raise an exception which will bubble up to the waiter.

Parameters

- **instance** – The instance for which an event is expected
- **event_names** – A list of event names. Each element can be a string event name or tuple of strings to indicate (name, tag).
- **deadline** – Maximum number of seconds we should wait for all of the specified events to arrive.
- **error_callback** – A function to be called if an event arrives

class InstanceEvents
Bases: object

cancel_all_events()

clear_events_for_instance(*instance*)
Remove all pending events for an instance.

This will remove all events currently pending for an instance and return them (indexed by event name).

Parameters **instance** – the instance for which events should be purged

Returns a dictionary of {event_name: eventlet.event.Event}

pop_instance_event(*instance*, *event*)
Remove a pending event from the wait list.

This will remove a pending event from the wait list so that it can be used to signal the waiters to wake up.

Parameters

- **instance** – the instance for which the event was generated
- **event** – the nova.objects.external_event.InstanceExternalEvent that describes the event

Returns the eventlet.event.Event object on which the waiters are blocked

prepare_for_instance_event(*instance*, *event_name*)
Prepare to receive an event for an instance.

This will register an event for the given instance that we will wait on later. This should be called before initiating whatever action will trigger the event. The resulting eventlet.event.Event object should be wait()'d on to ensure completion.

Parameters

- **instance** – the instance for which the event will be generated
- **event_name** – the name of the event we're expecting

Returns an event object that should be wait()'d on

delete_image_on_error(*f*)
Used for snapshot related method to ensure the image created in compute.api is deleted when an error occurs.

errors_out_migration(*f*)
Decorator to error out migration on failure.

object_compat(*function*)
Wraps a method that expects a new-world instance

This provides compatibility for callers passing old-style dict instances.

reverts_task_state(*f*)
Decorator to revert task_state on failure.

wrap_instance_event(*f*)
Wraps a method to log the event taken on the instance, and result.

This decorator wraps a method to log the start and result of an event, as part of an action taken on an instance.

wrap_instance_fault(*f*)
Wraps a method to catch exceptions related to instances.

This decorator wraps a method to catch any exceptions having to do with an instance that may get thrown. It then logs an instance fault in the db.

3.7.318 The nova.compute.monitors.base Module

class CPUMonitorBase(*compute_manager*)
Bases: nova.compute.monitors.base.MonitorBase

Base class for all monitors that return CPU-related metrics.

get_metric_names()

class MonitorBase(*compute_manager*)
Bases: object

Base class for all resource monitor plugins.

add_metrics_to_list(*metrics_list*)
Adds metric objects to a supplied list object.

Parameters **metric_list** – nova.objects.MonitorMetricList that the monitor plugin should append nova.objects.MonitorMetric objects to.

get_metric(*name*)
Return a (value, timestamp) tuple for the supplied metric name.

Parameters **name** – The name/key for the metric to grab the value for.

get_metric_names()
Get available metric names.

Get available metric names, which are represented by a set of keys that can be used to check conflicts and duplications :returns: set containing one or more values from

nova.objects.fields.MonitorMetricType.ALL constants

3.7.319 The nova.compute.monitors.cpu.virt_driver Module

CPU monitor based on virt driver to retrieve CPU information

class Monitor(*compute_manager*)
Bases: nova.compute.monitors.base.CPUMonitorBase

CPU monitor that uses the virt driver's get_host_cpu_stats() call.

get_metric(*name*)

3.7.320 The nova.compute.opts Module

list_opts()

3.7.321 The nova.compute.power_state Module

Power state is the state we get by calling virt driver on a particular domain. The hypervisor is always considered the authority on the status of a particular VM, and the power_state in the DB should be viewed as a snapshot of the VMs's state in the (recent) past. It can be periodically updated, and should also be updated at the end of a task if the task is supposed to affect power_state.

3.7.322 The nova.compute.resource_tracker Module

Track resources like memory and disk for a compute host. Provides the scheduler with useful information about availability through the ComputeNode model.

class ResourceTracker(*host*, *driver*, *nodename*)
Bases: object

Compute helper class for keeping track of resource usage as instances are built and destroyed.

abort_instance_claim(**args*, ***kwargs*)
Remove usage from the given instance.

disabled

drop_move_claim(**args*, ***kwargs*)
Remove usage for an incoming/outgoing migration.

instance_claim(**args*, ***kwargs*)
Indicate that some resources are needed for an upcoming compute instance build operation.

This should be called before the compute node is about to perform an instance build operation that will consume additional resources.

Parameters

- **context** – security context
- **instance_ref** (*nova.objects.instance.Instance object*) – instance to reserve resources for.
- **limits** – Dict of oversubscription limits for memory, disk, and CPUs.

Returns A Claim ticket representing the reserved resources. It can be used to revert the resource usage if an error occurs during the instance build.

resize_claim(**args*, ***kwargs*)
Indicate that resources are needed for a resize operation to this compute host. :param context: security context :param instance: instance object to reserve resources for :param instance_type: new instance_type being resized to :param limits: Dict of oversubscription limits for memory, disk, and CPUs :returns: A Claim ticket representing the reserved resources. This should be turned into finalize a resource claim or free resources after the compute operation is finished.

update_available_resource(*context*)
Override in-memory calculations of compute node resource usage based on data audited from the hypervisor layer.

Add in resource claims in progress to account for operations that have declared a need for resources, but not necessarily retrieved them from the hypervisor layer yet.

update_usage(**args*, ***kwargs*)
Update the resource usage and stats after a change in an instance

3.7.323 The nova.compute.resources.base Module

class Resource
Bases: object

This base class defines the interface used for compute resource plugins. It is not necessary to use this base class, but all compute resource plugins must implement the abstract methods found here. An instance of the plugin object is instantiated when it is loaded by calling __init__() with no parameters.

add_instance(*usage*)
Update resource information adding allocation according to the given resource usage.

Parameters **usage** – the resource usage of the instance being added

Returns None

remove_instance(*usage*)
Update resource information removing allocation according to the given resource usage.

Parameters **usage** – the resource usage of the instance being removed

Returns None

report_free()
Log free resources.

This method logs how much free resource is held by the resource plugin.

Returns None

reset(*resources*, *driver*)
Set the resource to an initial state based on the resource view discovered from the hypervisor.

test(*usage*, *limits*)
Test to see if we have sufficient resources to allocate for an instance with the given resource usage.

Parameters

- **usage** – the resource usage of the instances
- **limits** – limits to apply

Returns None if the test passes or a string describing the reason why the test failed

write(*resources*)

Write resource data to populate resources.

Parameters **resources** – the resources data to be populated

Returns None

3.7.324 The nova.compute.resources.vcpu Module

class VCPU

Bases: nova.compute.resources.base.Resource

VCPU compute resource plugin.

This is effectively a simple counter based on the vcpu requirement of each instance.

add_instance(*usage*)

remove_instance(*usage*)

report_free()

reset(*resources*, *driver*)

test(*usage*, *limits*)

write(*resources*)

3.7.325 The nova.compute.rpcapi Module

Client side of the compute RPC API.

class ComputeAPI

Bases: object

Client side of the compute rpc API.

API version history:

- 1.0 - Initial version.
- 1.1 - Adds get_host_uptime()
- 1.2 - Adds check_can_live_migrate_[destination|source]
- 1.3 - Adds change_instance_metadata()
- **1.4 - Remove instance_uuid, add instance argument to** reboot_instance()
- **1.5 - Remove instance_uuid, add instance argument to** pause_instance(), unpause_instance()
- **1.6 - Remove instance_uuid, add instance argument to** suspend_instance()
- **1.7 - Remove instance_uuid, add instance argument to** get_console_output()
- **1.8 - Remove instance_uuid, add instance argument to** add_fixed_ip_to_instance()

- 1.9 - Remove instance_uuid, add instance argument to attach_volume()
- **1.10 - Remove instance_id, add instance argument to** check_can_live_migrate_destination()
- **1.11 - Remove instance_id, add instance argument to** check_can_live_migrate_source()
- **1.12 - Remove instance_uuid, add instance argument to** confirm_resize()
- 1.13 - Remove instance_uuid, add instance argument to detach_volume()
- 1.14 - Remove instance_uuid, add instance argument to finish_resize()
- **1.15 - Remove instance_uuid, add instance argument to** finish_revert_resize()
- **1.16 - Remove instance_uuid, add instance argument to** get_diagnostics()
- **1.17 - Remove instance_uuid, add instance argument to** get_vnc_console()
- 1.18 - Remove instance_uuid, add instance argument to inject_file()
- **1.19 - Remove instance_uuid, add instance argument to** inject_network_info()
- **1.20 - Remove instance_id, add instance argument to** post_live_migration_at_destination()
- **1.21 - Remove instance_uuid, add instance argument to** power_off_instance() and stop_instance()
- **1.22 - Remove instance_uuid, add instance argument to** power_on_instance() and start_instance()
- **1.23 - Remove instance_id, add instance argument to** pre_live_migration()
- **1.24 - Remove instance_uuid, add instance argument to** rebuild_instance()
- **1.25 - Remove instance_uuid, add instance argument to** remove_fixed_ip_from_instance()
- **1.26 - Remove instance_id, add instance argument to** remove_volume_connection()
- **1.27 - Remove instance_uuid, add instance argument to** rescue_instance()
- 1.28 - Remove instance_uuid, add instance argument to reset_network()
- **1.29 - Remove instance_uuid, add instance argument to** resize_instance()
- **1.30 - Remove instance_uuid, add instance argument to** resume_instance()
- 1.31 - Remove instance_uuid, add instance argument to revert_resize()
- **1.32 - Remove instance_id, add instance argument to** rollback_live_migration_at_destination()
- **1.33 - Remove instance_uuid, add instance argument to** set_admin_password()
- **1.34 - Remove instance_uuid, add instance argument to** snapshot_instance()
- **1.35 - Remove instance_uuid, add instance argument to** unrescue_instance()
- **1.36 - Remove instance_uuid, add instance argument to** change_instance_metadata()
- **1.37 - Remove instance_uuid, add instance argument to** terminate_instance()
- **1.38 - Changes to prep_resize():**
  - remove instance_uuid, add instance
  - remove instance_type_id, add instance_type
  - remove topic, it was unused
- 1.39 - Remove instance_uuid, add instance argument to run_instance()
- 1.40 - Remove instance_id, add instance argument to live_migration()

- 1.41 - Adds refresh_instance_security_rules()
- **1.42 - Add reservations arg to prep_resize(), resize_instance(),** finish_resize(), confirm_resize(), revert_resize() and finish_revert_resize()
- 1.43 - Add migrate_data to live_migration()
- 1.44 - Adds reserve_block_device_name()
- 2.0 - Remove 1.x backwards compat
- 2.1 - Adds orig_sys_metadata to rebuild_instance()
- **2.2 - Adds slave_info parameter to add_aggregate_host() and** remove_aggregate_host()
- 2.3 - Adds volume_id to reserve_block_device_name()
- 2.4 - Add bdms to terminate_instance
- 2.5 - Add block device and network info to reboot_instance
- 2.6 - Remove migration_id, add migration to resize_instance
- 2.7 - Remove migration_id, add migration to confirm_resize
- 2.8 - Remove migration_id, add migration to finish_resize
- 2.9 - Add publish_service_capabilities()
- 2.10 - Adds filter_properties and request_spec to prep_resize()
- 2.11 - Adds soft_delete_instance() and restore_instance()
- 2.12 - Remove migration_id, add migration to revert_resize
- 2.13 - Remove migration_id, add migration to finish_revert_resize
- 2.14 - Remove aggregate_id, add aggregate to add_aggregate_host
- 2.15 - Remove aggregate_id, add aggregate to remove_aggregate_host
- 2.16 - Add instance_type to resize_instance
- 2.17 - Add get_backdoor_port()
- 2.18 - Add bdms to rebuild_instance
- 2.19 - Add node to run_instance
- 2.20 - Add node to prep_resize
- 2.21 - Add migrate_data dict param to pre_live_migration()
- **2.22 - Add recreate, on_shared_storage and host arguments to** rebuild_instance()
- 2.23 - Remove network_info from reboot_instance
- 2.24 - Added get_spice_console method
- 2.25 - Add attach_interface() and detach_interface()
- **2.26 - Add validate_console_port to ensure the service connects to** vnc on the correct port
- **2.27 - Adds 'reservations' to terminate_instance() and** soft_delete_instance()

... Grizzly supports message version 2.27. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.27.

- 2.28 - Adds check_instance_shared_storage()
- **2.29 - Made start_instance() and stop_instance() take new-world** instance objects

- 2.30 - Adds live_snapshot_instance()
- **2.31 - Adds shelve_instance(), shelve_offload_instance, and** unshelve_instance()
- 2.32 - Make reboot_instance take a new world instance object
- **2.33 - Made suspend_instance() and resume_instance() take new-world** instance objects
- 2.34 - Added swap_volume()
- **2.35 - Made terminate_instance() and soft_delete_instance() take** new-world instance objects
- **2.36 - Made pause_instance() and unpause_instance() take new-world** instance objects
- 2.37 - Added the legacy_bdm_in_spec parameter to run_instance
- **2.38 - Made check_can_live_migrate_[destination|source] take** new-world instance objects
- **2.39 - Made revert_resize() and confirm_resize() take new-world** instance objects
- 2.40 - Made reset_network() take new-world instance object
- 2.41 - Make inject_network_info take new-world instance object
- **2.42 - Splits snapshot_instance() into snapshot_instance() and** backup_instance() and makes them take new-world instance objects.
- 2.43 - Made prep_resize() take new-world instance object
- 2.44 - Add volume_snapshot_create(), volume_snapshot_delete()
- 2.45 - Made resize_instance() take new-world objects
- 2.46 - Made finish_resize() take new-world objects
- 2.47 - Made finish_revert_resize() take new-world objects

... Havana supports message version 2.47. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.47.

- 2.48 - Make add_aggregate_host() and remove_aggregate_host() take new-world objects
- ... - Remove live_snapshot() that was never actually used
- 3.0 - Remove 2.x compatibility
- 3.1 - Update get_spice_console() to take an instance object
- 3.2 - Update get_vnc_console() to take an instance object
- 3.3 - Update validate_console_port() to take an instance object
- 3.4 - Update rebuild_instance() to take an instance object
- 3.5 - Pass preserve_ephemeral flag to rebuild_instance()
- 3.6 - Make volume_snapshot_{create,delete} use new-world objects
- 3.7 - Update change_instance_metadata() to take an instance object
- 3.8 - Update set_admin_password() to take an instance object
- 3.9 - Update rescue_instance() to take an instance object
- 3.10 - Added get_rdp_console method
- 3.11 - Update unrescue_instance() to take an object
- 3.12 - Update add_fixed_ip_to_instance() to take an object

- 3.13 - Update remove_fixed_ip_from_instance() to take an object
- 3.14 - Update post_live_migration_at_destination() to take an object
- 3.15 - Adds filter_properties and node to unshelve_instance()
- **3.16 - Make reserve_block_device_name and attach_volume use new-world** objects, and add disk_bus and device_type params to reserve_block_device_name, and bdm param to attach_volume
- 3.17 - Update attach_interface and detach_interface to take an object
- **3.18 - Update get_diagnostics() to take an instance object**
  - Removed inject_file(), as it was unused.
- 3.19 - Update pre_live_migration to take instance object
- 3.20 - Make restore_instance take an instance object
- 3.21 - Made rebuild take new-world BDM objects
- 3.22 - Made terminate_instance take new-world BDM objects
- **3.23 - Added external_instance_event()**
  - build_and_run_instance was added in Havana and not used or documented.

... Icehouse supports message version 3.23. So, any changes to existing methods in 3.x after that point should be done such that they can handle the version_cap being set to 3.23.

- 3.24 - Update rescue_instance() to take optional rescue_image_ref
- 3.25 - Make detach_volume take an object
- 3.26 - Make live_migration() and rollback_live_migration_at_destination() take an object
- ... Removed run_instance()
- 3.27 - Make run_instance() accept a new-world object
- 3.28 - Update get_console_output() to accept a new-world object
- 3.29 - Make check_instance_shared_storage accept a new-world object
- 3.30 - Make remove_volume_connection() accept a new-world object
- 3.31 - Add get_instance_diagnostics
- **3.32 - Add destroy_disks and migrate_data optional parameters to** rollback_live_migration_at_destination()
- 3.33 - Make build_and_run_instance() take a NetworkRequestList object
- 3.34 - Add get_serial_console method
- 3.35 - Make reserve_block_device_name return a BDM object

... Juno supports message version 3.35. So, any changes to existing methods in 3.x after that point should be done such that they can handle the version_cap being set to 3.35.

- 3.36 - Make build_and_run_instance() send a Flavor object
- **3.37 - Add clean_shutdown to stop, resize, rescue, shelve, and** shelve_offload
- 3.38 - Add clean_shutdown to prep_resize
- 3.39 - Add quiesce_instance and unquiesce_instance methods
- **3.40 - Make build_and_run_instance() take a new-world topology** limits object

... Kilo supports messaging version 3.40. So, any changes to existing methods in 3.x after that point should be done so that they can handle the version_cap being set to 3.40

... Version 4.0 is equivalent to 3.40. Kilo sends version 4.0 by default, can accept 3.x calls from Juno nodes, and can be pinned to 3.x for Juno compatibility. All new changes should go against 4.x.

- 4.0 - Remove 3.x compatibility
- 4.1 - Make prep_resize() and resize_instance() send Flavor object
- 4.2 - Add migration argument to live_migration()

VERSION_ALIASES = {'icehouse': '3.23', 'juno': '3.35', 'kilo': '4.0'}

add_aggregate_host(*ctxt*, *aggregate*, *host_param*, *host*, *slave_info=None*)
Add aggregate host.

Parameters

- **ctxt** – request context
- **aggregate** –
- **host_param** – This value is placed in the message to be the 'host' parameter for the remote method.
- **host** – This is the host to send the message to.

add_fixed_ip_to_instance(*ctxt*, *instance*, *network_id*)

attach_interface(*ctxt*, *instance*, *network_id*, *port_id*, *requested_ip*)

attach_volume(*ctxt*, *instance*, *bdm*)

backup_instance(*ctxt*, *instance*, *image_id*, *backup_type*, *rotation*)

build_and_run_instance(*ctxt*, *instance*, *host*, *image*, *request_spec*, *filter_properties*, *admin_password=None*, *injected_files=None*, *requested_networks=None*, *security_groups=None*, *block_device_mapping=None*, *node=None*, *limits=None*)

change_instance_metadata(*ctxt*, *instance*, *diff*)

check_can_live_migrate_destination(*ctxt*, *instance*, *destination*, *block_migration*, *disk_over_commit*)

check_can_live_migrate_source(*ctxt*, *instance*, *dest_check_data*)

check_instance_shared_storage(*ctxt*, *instance*, *data*, *host=None*)

confirm_resize(*ctxt*, *instance*, *migration*, *host*, *reservations=None*, *cast=True*)

detach_interface(*ctxt*, *instance*, *port_id*)

detach_volume(*ctxt*, *instance*, *volume_id*)

external_instance_event(*ctxt*, *instances*, *events*)

finish_resize(*ctxt*, *instance*, *migration*, *image*, *disk_info*, *host*, *reservations=None*)

finish_revert_resize(*ctxt*, *instance*, *migration*, *host*, *reservations=None*)

get_client(*target*, *version_cap*, *serializer*)

get_console_output(*ctxt*, *instance*, *tail_length*)

get_console_pool_info(*ctxt*, *console_type*, *host*)

get_console_topic(*ctxt*, *host*)

get_diagnostics(*ctxt*, *instance*)

get_host_uptime(*ctxt*, *host*)

get_instance_diagnostics(*ctxt*, *instance*)

get_rdp_console(*ctxt*, *instance*, *console_type*)

get_serial_console(*ctxt*, *instance*, *console_type*)

get_spice_console(*ctxt*, *instance*, *console_type*)

get_vnc_console(*ctxt*, *instance*, *console_type*)

host_maintenance_mode(*ctxt*, *host_param*, *mode*, *host*)
Set host maintenance mode

Parameters

- **ctxt** – request context
- **host_param** – This value is placed in the message to be the 'host' parameter for the remote method.
- **mode** –
- **host** – This is the host to send the message to.

host_power_action(*ctxt*, *action*, *host*)

inject_network_info(*ctxt*, *instance*)

live_migration(*ctxt*, *instance*, *dest*, *block_migration*, *host*, *migration*, *migrate_data=None*)

pause_instance(*ctxt*, *instance*)

post_live_migration_at_destination(*ctxt*, *instance*, *block_migration*, *host*)

pre_live_migration(*ctxt*, *instance*, *block_migration*, *disk*, *host*, *migrate_data=None*)

prep_resize(*ctxt*, *image*, *instance*, *instance_type*, *host*, *reservations=None*, *request_spec=None*, *filter_properties=None*, *node=None*, *clean_shutdown=True*)

quiesce_instance(*ctxt*, *instance*)

reboot_instance(*ctxt*, *instance*, *block_device_info*, *reboot_type*)

rebuild_instance(*ctxt*, *instance*, *new_pass*, *injected_files*, *image_ref*, *orig_image_ref*, *orig_sys_metadata*, *bdms*, *recreate=False*, *on_shared_storage=False*, *host=None*, *preserve_ephemeral=False*, *kwargs=None*)

refresh_instance_security_rules(*ctxt*, *host*, *instance*)

refresh_provider_fw_rules(*ctxt*, *host*)

refresh_security_group_members(*ctxt*, *security_group_id*, *host*)

refresh_security_group_rules(*ctxt*, *security_group_id*, *host*)

remove_aggregate_host(*ctxt*, *aggregate*, *host_param*, *host*, *slave_info=None*)
Remove aggregate host.

Parameters

- **ctxt** – request context
- **aggregate** –
- **host_param** – This value is placed in the message to be the 'host' parameter for the remote method.

- **host** – This is the host to send the message to.

remove_fixed_ip_from_instance(*ctxt*, *instance*, *address*)

remove_volume_connection(*ctxt*, *instance*, *volume_id*, *host*)

rescue_instance(*ctxt*, *instance*, *rescue_password*, *rescue_image_ref=None*, *clean_shutdown=True*)

reserve_block_device_name(*ctxt*, *instance*, *device*, *volume_id*, *disk_bus=None*, *device_type=None*)

reset_network(*ctxt*, *instance*)

resize_instance(*ctxt*, *instance*, *migration*, *image*, *instance_type*, *reservations=None*, *clean_shutdown=True*)

restore_instance(*ctxt*, *instance*)

resume_instance(*ctxt*, *instance*)

revert_resize(*ctxt*, *instance*, *migration*, *host*, *reservations=None*)

rollback_live_migration_at_destination(*ctxt*, *instance*, *host*, *destroy_disks=True*, *migrate_data=None*)

set_admin_password(*ctxt*, *instance*, *new_pass*)

set_host_enabled(*ctxt*, *enabled*, *host*)

shelve_instance(*ctxt*, *instance*, *image_id=None*, *clean_shutdown=True*)

shelve_offload_instance(*ctxt*, *instance*, *clean_shutdown=True*)

snapshot_instance(*ctxt*, *instance*, *image_id*)

soft_delete_instance(*ctxt*, *instance*, *reservations=None*)

start_instance(*ctxt*, *instance*)

stop_instance(*ctxt*, *instance*, *do_cast=True*, *clean_shutdown=True*)

suspend_instance(*ctxt*, *instance*)

swap_volume(*ctxt*, *instance*, *old_volume_id*, *new_volume_id*)

terminate_instance(*ctxt*, *instance*, *bdms*, *reservations=None*)

unpause_instance(*ctxt*, *instance*)

unquiesce_instance(*ctxt*, *instance*, *mapping=None*)

unrescue_instance(*ctxt*, *instance*)

unshelve_instance(*ctxt*, *instance*, *host*, *image=None*, *filter_properties=None*, *node=None*)

validate_console_port(*ctxt*, *instance*, *port*, *console_type*)

volume_snapshot_create(*ctxt*, *instance*, *volume_id*, *create_info*)

volume_snapshot_delete(*ctxt*, *instance*, *volume_id*, *snapshot_id*, *delete_info*)

3.7.326 The `nova.compute.stats` Module

class Stats

Bases: `dict`

Handler for updates to compute node workload stats.

calculate_workload()
Calculate current load of the compute host based on task states.

clear()

digest_stats(*stats*)
Apply stats provided as a dict or a json encoded string.

io_workload
Calculate an I/O based load by counting I/O heavy operations.

num_instances

num_instances_for_project(*project_id*)

num_os_type(*os_type*)

update_stats_for_instance(*instance*)
Update stats after an instance is changed.

3.7.327 The nova.compute.task_states Module

Possible task states for instances.

Compute instance task states represent what is happening to the instance at the current moment. These tasks can be generic, such as 'spawning', or specific, such as 'block_device_mapping'. These task states allow for a better view into what an instance is doing and should be displayed to users/administrators as necessary.

3.7.328 The nova.compute.utils Module

Compute-related Utilities and helpers.

class EventReporter(*context*, *event_name*, **instance_uuids*)
Bases: object

Context manager to report instance action events.

class UnlimitedSemaphore
Bases: object

balance

add_instance_fault_from_exc(*context*, *instance*, *fault*, *exc_info=None*)
Adds the specified fault to the database.

default_device_names_for_instance(*instance*, *root_device_name*, **block_device_lists*)
Generate missing device names for an instance.

exception_to_dict(*fault*)
Converts exceptions to a dict for use in notifications.

finish_instance_usage_audit(*context*, *conductor*, *begin*, *end*, *host*, *errors*, *message*)

get_device_name_for_instance(*instance*, *bdms*, *device*)
Validates (or generates) a device name for instance.

This method is a wrapper for get_next_device_name that gets the list of used devices and the root device from a block device mapping.

get_machine_ips()
Get the machine's ip addresses

Returns list of Strings of ip addresses

get_next_device_name(*instance*, *device_name_list*, *root_device_name=None*, *device=None*)
Validates (or generates) a device name for instance.

If device is not set, it will generate a unique device appropriate for the instance. It uses the root_device_name (if provided) and the list of used devices to find valid device names. If the device name is valid but applicable to a different backend (for example /dev/vdc is specified but the backend uses /dev/xvdc), the device name will be converted to the appropriate format.

get_nw_info_for_instance(*instance*)

get_reboot_type(*task_state*, *current_power_state*)
Checks if the current instance state requires a HARD reboot.

get_value_from_system_metadata(*instance*, *key*, *type*, *default*)
Get a value of a specified type from image metadata.

@param instance: The instance object @param key: The name of the property to get @param type: The python type the value is be returned as @param default: The value to return if key is not set or not the right type

has_audit_been_run(*context*, *conductor*, *host*, *timestamp=None*)

notify_about_aggregate_update(*context*, *event_suffix*, *aggregate_payload*)
Send a notification about aggregate update.

Parameters

- **event_suffix** – Event type like "create.start" or "create.end"
- **aggregate_payload** – payload for aggregate update

notify_about_host_update(*context*, *event_suffix*, *host_payload*)
Send a notification about host update.

Parameters

- **event_suffix** – Event type like "create.start" or "create.end"
- **host_payload** – payload for host update. It is a dict and there should be at least the 'host_name' key in this dict.

notify_about_instance_usage(*notifier*, *context*, *instance*, *event_suffix*, *network_info=None*, *system_metadata=None*, *extra_usage_info=None*, *fault=None*)
Send a notification about an instance.

Parameters

- **notifier** – a messaging.Notifier
- **event_suffix** – Event type like "delete.start" or "exists"
- **network_info** – Networking information, if provided.
- **system_metadata** – system_metadata DB entries for the instance, if provided.
- **extra_usage_info** – Dictionary containing extra values to add or override in the notification.

notify_about_server_group_update(*context*, *event_suffix*, *sg_payload*)
Send a notification about server group update.

Parameters

- **event_suffix** – Event type like "create.start" or "create.end"
- **sg_payload** – payload for server group update

notify_usage_exists(*notifier*, *context*, *instance_ref*, *current_period=False*, *ignore_missing_network_data=True*, *system_metadata=None*, *extra_usage_info=None*)

Generates 'exists' notification for an instance for usage auditing purposes.

Parameters

- **notifier** – a messaging.Notifier
- **current_period** – if True, this will generate a usage for the current usage period; if False, this will generate a usage for the previous audit period.
- **ignore_missing_network_data** – if True, log any exceptions generated while getting network info; if False, raise the exception.
- **system_metadata** – system_metadata DB entries for the instance, if not None. *NOTE*: Currently unused here in trunk, but needed for potential custom modifications.
- **extra_usage_info** – Dictionary containing extra values to add or override in the notification if not None.

remove_shelved_keys_from_system_metadata(*instance*)

start_instance_usage_audit(*context*, *conductor*, *begin*, *end*, *host*, *num_instances*)

usage_volume_info(*vol_usage*)

3.7.329 The nova.compute.vm_mode Module

Possible vm modes for instances.

Compute instance vm modes represent the host/guest ABI used for the virtual machine / container. Individual hypervisors may support multiple different vm modes per host. Available vm modes for a hypervisor driver may also vary according to the architecture it is running on.

The 'vm_mode' parameter can be set against an instance to choose what sort of VM to boot.

canonicalize(*mode*)

Canonicalize the vm mode

Parameters **name** – vm mode name to canonicalize

Returns a canonical vm mode name

get_from_instance(*instance*)

Get the vm mode for an instance

Parameters **instance** – instance object to query

Returns canonicalized vm mode for the instance

is_valid(*name*)

Check if a string is a valid vm mode

Parameters **name** – vm mode name to validate

Returns True if @name is valid

3.7.330 The nova.compute.vm_states Module

Possible vm states for instances.

Compute instance vm states represent the state of an instance as it pertains to a user or administrator.

vm_state describes a VM's current stable (not transition) state. That is, if there is no ongoing compute API calls (running tasks), vm_state should reflect what the customer expect the VM to be. When combined with task states (task_states.py), a better picture can be formed regarding the instance's health and progress.

See http://wiki.openstack.org/VMState

3.7.331 The `nova.conductor.api` Module

Handles all requests to the conductor service.

class API

Bases: `nova.conductor.api.LocalAPI`

Conductor API that does updates via RPC to the ConductorManager.

instance_update(*context*, *instance_uuid*, ***updates*)
Perform an instance update in the database.

wait_until_ready(*context*, *early_timeout=10*, *early_attempts=10*)
Wait until a conductor service is up and running.

This method calls the remote ping() method on the conductor topic until it gets a response. It starts with a shorter timeout in the loop (early_timeout) up to early_attempts number of tries. It then drops back to the globally configured timeout for rpc calls for each retry.

class ComputeTaskAPI

Bases: `object`

ComputeTask API that queues up compute tasks for nova-conductor.

build_instances(*context*, *instances*, *image*, *filter_properties*, *admin_password*, *injected_files*, *requested_networks*, *security_groups*, *block_device_mapping*, *legacy_bdm=True*)

live_migrate_instance(*context*, *instance*, *host_name*, *block_migration*, *disk_over_commit*)

rebuild_instance(*context*, *instance*, *orig_image_ref*, *image_ref*, *injected_files*, *new_pass*, *orig_sys_metadata*, *bdms*, *recreate=False*, *on_shared_storage=False*, *preserve_ephemeral=False*, *host=None*, *kwargs=None*)

resize_instance(*context*, *instance*, *extra_instance_updates*, *scheduler_hint*, *flavor*, *reservations*, *clean_shutdown=True*)

unshelve_instance(*context*, *instance*)

class LocalAPI

Bases: `object`

A local version of the conductor API that does database updates locally instead of via RPC.

compute_node_create(*context*, *values*)

instance_update(*context*, *instance_uuid*, ***updates*)
Perform an instance update in the database.

object_backport(*context*, *objinst*, *target_version*)

provider_fw_rule_get_all(*context*)

security_groups_trigger_members_refresh(*context*, *group_ids*)

task_log_begin_task(*context*, *task_name*, *begin*, *end*, *host*, *task_items=None*, *message=None*)

task_log_end_task(*context*, *task_name*, *begin*, *end*, *host*, *errors*, *message=None*)

task_log_get(*context*, *task_name*, *begin*, *end*, *host*, *state=None*)

vol_usage_update(*context*, *vol_id*, *rd_req*, *rd_bytes*, *wr_req*, *wr_bytes*, *instance*, *last_refreshed=None*, *update_totals=False*)

wait_until_ready(*context*, **args*, ***kwargs*)

class LocalComputeTaskAPI

Bases: object

build_instances(*context*, *instances*, *image*, *filter_properties*, *admin_password*, *injected_files*, *requested_networks*, *security_groups*, *block_device_mapping*, *legacy_bdm=True*)

live_migrate_instance(*context*, *instance*, *host_name*, *block_migration*, *disk_over_commit*)

rebuild_instance(*context*, *instance*, *orig_image_ref*, *image_ref*, *injected_files*, *new_pass*, *orig_sys_metadata*, *bdms*, *recreate=False*, *on_shared_storage=False*, *preserve_ephemeral=False*, *host=None*, *kwargs=None*)

resize_instance(*context*, *instance*, *extra_instance_updates*, *scheduler_hint*, *flavor*, *reservations*, *clean_shutdown=True*)

unshelve_instance(*context*, *instance*)

3.7.332 The nova.conductor.manager Module

Handles database requests from other nova services.

class ComputeTaskManager

Bases: nova.db.base.Base

Namespace for compute methods.

This class presents an rpc API for nova-conductor under the 'compute_task' namespace. The methods here are compute operations that are invoked by the API service. These methods see the operation to completion, which may involve coordinating activities on multiple compute nodes.

build_instances(*context*, *instances*, *image*, *filter_properties*, *admin_password*, *injected_files*, *requested_networks*, *security_groups*, *block_device_mapping=None*, *legacy_bdm=True*)

migrate_server(**args*, ***kwargs*)

rebuild_instance(*context*, *instance*, *orig_image_ref*, *image_ref*, *injected_files*, *new_pass*, *orig_sys_metadata*, *bdms*, *recreate*, *on_shared_storage*, *preserve_ephemeral=False*, *host=None*)

target = <Target namespace=compute_task, version=1.11>

unshelve_instance(*context*, *instance*)

class ConductorManager(**args*, ***kwargs*)

Bases: nova.manager.Manager

Mission: Conduct things.

The methods in the base API for nova-conductor are various proxy operations performed on behalf of the nova-compute service running on compute nodes. Compute nodes are not allowed to directly access the database, so this set of methods allows them to get specific work done without locally accessing the database.

The nova-conductor service also exposes an API in the 'compute_task' namespace. See the ComputeTaskManager class for details.

action_event_finish(**args*, ***kwargs*)

action_event_start(**args*, ***kwargs*)

agent_build_get_by_triple(*context*, *hypervisor*, *os*, *architecture*)

aggregate_host_add(**args*, ***kwargs*)

aggregate_host_delete(**args*, ***kwargs*)

aggregate_metadata_get_by_host(*context*, *host*, *key='availability_zone'*)

block_device_mapping_get_all_by_instance(*context*, *instance*, *legacy*)

block_device_mapping_update_or_create(*context*, *values*, *create*)

bw_usage_update(*context*, *uuid*, *mac*, *start_period*, *bw_in*, *bw_out*, *last_ctr_in*, *last_ctr_out*, *last_refreshed*, *update_cells*)

compute_api

compute_node_create(*context*, *values*)

compute_node_delete(*context*, *node*)

compute_node_update(*context*, *node*, *values*)

compute_unrescue(*context*, *instance*)

get_ec2_ids(*context*, *instance*)

instance_destroy(*context*, *instance*)

instance_fault_create(*context*, *values*)

instance_get_active_by_window_joined(*context*, *begin*, *end*, *project_id*, *host*)

instance_get_all_by_filters(*context*, *filters*, *sort_key*, *sort_dir*, *columns_to_join*, *use_slave*)

instance_get_all_by_host(*context*, *host*, *node*, *columns_to_join*)

instance_get_by_uuid(**args*, ***kwargs*)

instance_update(**args*, ***kwargs*)

migration_get_in_progress_by_host_and_node(*context*, *host*, *node*)

network_api

network_migrate_instance_finish(*context*, *instance*, *migration*)

network_migrate_instance_start(*context*, *instance*, *migration*)

notify_usage_exists(*context*, *instance*, *current_period*, *ignore_missing_network_data*, *system_metadata*, *extra_usage_info*)

object_action(*context*, *objinst*, *objmethod*, *args*, *kwargs*)
Perform an action on an object.

object_backport(*context*, *objinst*, *target_version*)

object_class_action(*context*, *objname*, *objmethod*, *objver*, *args*, *kwargs*)
Perform a classmethod action on an object.

provider_fw_rule_get_all(*context*)

quota_commit(*context*, *reservations*, *project_id=None*, *user_id=None*)

quota_rollback(*context*, *reservations*, *project_id=None*, *user_id=None*)

security_groups_trigger_handler(*context*, *event*, *args*)

security_groups_trigger_members_refresh(*context*, *group_ids*)

service_create(*context*, *values*)

service_destroy(**args*, ***kwargs*)

service_get_all_by(**args*, ***kwargs*)

service_update(**args*, ***kwargs*)

target = <Target version=2.1>

task_log_begin_task(*context*, *task_name*, *begin*, *end*, *host*, *task_items*, *message*)

task_log_end_task(*context*, *task_name*, *begin*, *end*, *host*, *errors*, *message*)

task_log_get(*context*, *task_name*, *begin*, *end*, *host*, *state*)

vol_usage_update(*context*, *vol_id*, *rd_req*, *rd_bytes*, *wr_req*, *wr_bytes*, *instance*, *last_refreshed*, *update_totals*)

3.7.333 The nova.conductor.rpcapi Module

Client side of the conductor RPC API.

class ComputeTaskAPI

Bases: object

Client side of the conductor 'compute' namespaced RPC API

API version history:

1.0 - Initial version (empty). 1.1 - Added unified migrate_server call. 1.2 - Added build_instances 1.3 - Added unshelve_instance 1.4 - Added reservations to migrate_server. 1.5 - Added the leagacy_bdm parameter to build_instances 1.6 - Made migrate_server use instance objects 1.7 - Do not send block_device_mapping and legacy_bdm to build_instances 1.8 - Add rebuild_instance 1.9 - Converted requested_networks to NetworkRequestList object 1.10 - Made migrate_server() and build_instances() send flavor objects 1.11 - Added clean_shutdown to migrate_server()

build_instances(*context*, *instances*, *image*, *filter_properties*, *admin_password*, *injected_files*, *requested_networks*, *security_groups*, *block_device_mapping*, *legacy_bdm=True*)

migrate_server(*context*, *instance*, *scheduler_hint*, *live*, *rebuild*, *flavor*, *block_migration*, *disk_over_commit*, *reservations=None*, *clean_shutdown=True*)

rebuild_instance(*ctxt*, *instance*, *new_pass*, *injected_files*, *image_ref*, *orig_image_ref*, *orig_sys_metadata*, *bdms*, *recreate=False*, *on_shared_storage=False*, *host=None*, *preserve_ephemeral=False*, *kwargs=None*)

unshelve_instance(*context*, *instance*)

class ConductorAPI

Bases: object

Client side of the conductor RPC API

API version history:

- 1.0 - Initial version.
- 1.1 - Added migration_update
- 1.2 - Added instance_get_by_uuid and instance_get_all_by_host
- 1.3 - Added aggregate_host_add and aggregate_host_delete
- 1.4 - Added migration_get
- 1.5 - Added bw_usage_update

- 1.6 - Added get_backdoor_port()
- 1.7 - Added aggregate_get_by_host, aggregate_metadata_add, and aggregate_metadata_delete
- 1.8 - Added security_group_get_by_instance and security_group_rule_get_by_security_group
- 1.9 - Added provider_fw_rule_get_all
- 1.10 - Added agent_build_get_by_triple
- 1.11 - Added aggregate_get
- 1.12 - Added block_device_mapping_update_or_create
- 1.13 - Added block_device_mapping_get_all_by_instance
- 1.14 - Added block_device_mapping_destroy
- 1.15 - Added instance_get_all_by_filters and instance_get_all_hung_in_rebooting and instance_get_active_by_window Deprecated instance_get_all_by_host
- 1.16 - Added instance_destroy
- 1.17 - Added instance_info_cache_delete
- 1.18 - Added instance_type_get
- 1.19 - Added vol_get_usage_by_time and vol_usage_update
- 1.20 - Added migration_get_unconfirmed_by_dest_compute
- 1.21 - Added service_get_all_by
- 1.22 - Added ping
- **1.23 - Added instance_get_all** Un-Deprecate instance_get_all_by_host
- 1.24 - Added instance_get
- 1.25 - Added action_event_start and action_event_finish
- 1.26 - Added instance_info_cache_update
- 1.27 - Added service_create
- 1.28 - Added binary arg to service_get_all_by
- 1.29 - Added service_destroy
- 1.30 - Added migration_create
- 1.31 - Added migration_get_in_progress_by_host_and_node
- 1.32 - Added optional node to instance_get_all_by_host
- 1.33 - Added compute_node_create and compute_node_update
- 1.34 - Added service_update
- 1.35 - Added instance_get_active_by_window_joined
- 1.36 - Added instance_fault_create
- 1.37 - Added task_log_get, task_log_begin_task, task_log_end_task
- 1.38 - Added service name to instance_update
- 1.39 - Added notify_usage_exists
- 1.40 - Added security_groups_trigger_handler and security_groups_trigger_members_refresh Remove instance_get_active_by_window

- 1.41 - Added fixed_ip_get_by_instance, network_get, instance_floating_address_get_all, quota_commit, quota_rollback
- 1.42 - Added get_ec2_ids, aggregate_metadata_get_by_host
- 1.43 - Added compute_stop
- 1.44 - Added compute_node_delete
- 1.45 - Added project_id to quota_commit and quota_rollback
- 1.46 - Added compute_confirm_resize
- 1.47 - Added columns_to_join to instance_get_all_by_host and instance_get_all_by_filters
- 1.48 - Added compute_unrescue

... Grizzly supports message version 1.48. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 1.48.

- 1.49 - Added columns_to_join to instance_get_by_uuid
- 1.50 - Added object_action() and object_class_action()
- **1.51 - Added the 'legacy' argument to** block_device_mapping_get_all_by_instance
- 1.52 - Pass instance objects for compute_confirm_resize
- 1.53 - Added compute_reboot
- 1.54 - Added 'update_cells' argument to bw_usage_update
- 1.55 - Pass instance objects for compute_stop
- **1.56 - Remove compute_confirm_resize and** migration_get_unconfirmed_by_dest_compute
- 1.57 - Remove migration_create()
- 1.58 - Remove migration_get()

... Havana supports message version 1.58. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.58.

- 1.59 - Remove instance_info_cache_update()
- 1.60 - Remove aggregate_metadata_add() and aggregate_metadata_delete()
- **... - Remove security_group_get_by_instance() and** security_group_rule_get_by_security_group()
- 1.61 - Return deleted instance from instance_destroy()
- 1.62 - Added object_backport()
- **1.63 - Changed the format of values['stats'] from a dict to a JSON string** in compute_node_update()
- **1.64 - Added use_slave to instance_get_all_filters()**
  - Remove instance_type_get()
  - Remove aggregate_get()
  - Remove aggregate_get_by_host()
  - Remove instance_get()
  - Remove migration_update()
  - Remove block_device_mapping_destroy()
- **2.0 - Drop backwards compatibility**

- Remove quota_rollback() and quota_commit()
- Remove aggregate_host_add() and aggregate_host_delete()
- Remove network_migrate_instance_start() and network_migrate_instance_finish()
- Remove vol_get_usage_by_time

... Icehouse supports message version 2.0. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.0.

•Remove instance_destroy()

•Remove compute_unrescue()

•Remove instance_get_all_by_filters()

•Remove instance_get_active_by_window_joined()

•Remove instance_fault_create()

•Remove action_event_start() and action_event_finish()

•Remove instance_get_by_uuid()

•Remove agent_build_get_by_triple()

... Juno supports message version 2.0. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.0.

•2.1 - Make notify_usage_exists() take an instance object

•Remove bw_usage_update()

•Remove notify_usage_exists()

... Kilo supports message version 2.1. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.1.

•Remove get_ec2_ids()

•Remove service_get_all_by()

•Remove service_create()

•Remove service_destroy()

•Remove service_update()

•Remove migration_get_in_progress_by_host_and_node()

•Remove aggregate_metadata_get_by_host()

•Remove block_device_mapping_update_or_create()

•Remove block_device_mapping_get_all_by_instance()

•Remove instance_get_all_by_host()

•Remove compute_node_update()

•Remove compute_node_delete()

•Remove security_groups_trigger_handler()

VERSION_ALIASES = {'kilo': '2.1', 'grizzly': '1.48', 'havana': '1.58', 'juno': '2.0', 'icehouse': '2.0'}

compute_node_create(*context*, *values*)

instance_update(*context*, *instance_uuid*, *updates*, *service=None*)

object_action(*context*, *objinst*, *objmethod*, *args*, *kwargs*)

object_backport(*context*, *objinst*, *target_version*)

object_class_action(*context*, *objname*, *objmethod*, *objver*, *args*, *kwargs*)

provider_fw_rule_get_all(*context*)

security_groups_trigger_members_refresh(*context*, *group_ids*)

task_log_begin_task(*context*, *task_name*, *begin*, *end*, *host*, *task_items=None*, *message=None*)

task_log_end_task(*context*, *task_name*, *begin*, *end*, *host*, *errors*, *message=None*)

task_log_get(*context*, *task_name*, *begin*, *end*, *host*, *state=None*)

vol_usage_update(*context*, *vol_id*, *rd_req*, *rd_bytes*, *wr_req*, *wr_bytes*, *instance*, *last_refreshed=None*, *update_totals=False*)

3.7.334 The nova.conductor.tasks.live_migrate Module

class LiveMigrationTask(*context*, *instance*, *destination*, *block_migration*, *disk_over_commit*, *migration*)

Bases: object

execute()

rollback()

execute(*context*, *instance*, *destination*, *block_migration*, *disk_over_commit*, *migration*)

3.7.335 The nova.config Module

parse_args(*argv*, *default_config_files=None*)

3.7.336 The nova.console.api Module

Handles ConsoleProxy API requests.

class API(***kwargs*)

Bases: nova.db.base.Base

API for spinning up or down console proxy connections.

create_console(*context*, *instance_uuid*)

delete_console(*context*, *instance_uuid*, *console_uuid*)

get_console(*context*, *instance_uuid*, *console_uuid*)

get_consoles(*context*, *instance_uuid*)

3.7.337 The nova.console.fake Module

Fake ConsoleProxy driver for tests.

class FakeConsoleProxy

Bases: object

Fake ConsoleProxy driver.

console_type

fix_console_password(*password*)
Trim password to length, and any other massaging.

fix_pool_password(*password*)
Trim password to length, and any other massaging.

generate_password(*length=8*)
Returns random console password.

get_port(*context*)
Get available port for consoles that need one.

init_host()
Start up any config'ed consoles on start.

setup_console(*context*, *console*)
Sets up actual proxies.

teardown_console(*context*, *console*)
Tears down actual proxies.

3.7.338 The nova.console.manager Module

Console Proxy Service.

class ConsoleProxyManager(*console_driver=None*, **args*, ***kwargs*)
Bases: nova.manager.Manager

Sets up and tears down any console proxy connections.

Needed for accessing instance consoles securely.

add_console(*context*, *instance_id*)

init_host()

remove_console(*context*, *console_id*)

target = <Target version=2.0>

3.7.339 The nova.console.rpcapi Module

Client side of the console RPC API.

class ConsoleAPI(*topic=None*, *server=None*)
Bases: object

Client side of the console rpc API.

API version history:

1.0 - Initial version. 1.1 - Added get_backdoor_port()

... Grizzly and Havana support message version 1.1. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.1.

2.0 - Major API rev for Icehouse

... Icehouse, Juno and Kilo support message version 2.0. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.0.

VERSION_ALIASES = {'kilo': '2.0', 'grizzly': '1.1', 'havana': '1.1', 'juno': '2.0', 'icehouse': '2.0'}

add_console(*ctxt*, *instance_id*)

remove_console(*ctxt*, *console_id*)

3.7.340 The nova.console.serial Module

Serial consoles module.

acquire_port(**args*, ***kwargs*)
Returns a free TCP port on host.

Find and returns a free TCP port on 'host' in the range of 'CONF.serial_console.port_range'.

release_port(**args*, ***kwargs*)
Release TCP port to be used next time.

3.7.341 The nova.console.type Module

class Console(*host*, *port*, *internal_access_path=None*)
Bases: object

get_connection_info(*token*, *access_url*)
Returns an unreferenced dict with connection information.

class ConsoleRDP(*host*, *port*, *internal_access_path=None*)
Bases: nova.console.type.Console

class ConsoleSerial(*host*, *port*, *internal_access_path=None*)
Bases: nova.console.type.Console

class ConsoleSpice(*host*, *port*, *tlsPort*, *internal_access_path=None*)
Bases: nova.console.type.Console

class ConsoleVNC(*host*, *port*, *internal_access_path=None*)
Bases: nova.console.type.Console

3.7.342 The nova.console.websocketproxy Module

Websocket proxy that is compatible with OpenStack Nova. Leverages websockify.py by Joel Martin

class NovaProxyRequestHandler(**args*, ***kwargs*)
Bases: nova.console.websocketproxy.NovaProxyRequestHandlerBase, websockify.websocketproxy.ProxyRequestHandler

socket(**args*, ***kwargs*)

class NovaProxyRequestHandlerBase
Bases: object

address_string()

new_websocket_client()
Called after a new WebSocket connection has been established.

verify_origin_proto(*connection_info*, *origin_proto*)

class NovaWebSocketProxy(*RequestHandlerClass=<class websockify.websocketproxy.ProxyRequestHandler at 0x7f8582a80b48>*, **args*, ***kwargs*)
Bases: websockify.websocketproxy.WebSocketProxy

static get_logger()

3.7.343 The nova.console.xvp Module

XVP (Xenserver VNC Proxy) driver.

class XVPConsoleProxy
Bases: object

Sets up XVP config, and manages XVP daemon.

console_type

fix_console_password(*password*)
Trim password to length, and encode.

fix_pool_password(*password*)
Trim password to length, and encode.

get_port(*context*)
Get available port for consoles that need one.

init_host()
Start up any config'ed consoles on start.

setup_console(*context*, *console*)
Sets up actual proxies.

teardown_console(*context*, *console*)
Tears down actual proxies.

3.7.344 The nova.consoleauth.manager Module

Auth Components for Consoles.

class ConsoleAuthManager(*scheduler_driver=None*, **args*, ***kwargs*)
Bases: nova.manager.Manager

Manages token based authentication.

authorize_console(*context*, *token*, *console_type*, *host*, *port*, *internal_access_path*, *instance_uuid*, *access_url=None*)

check_token(*context*, *token*)

delete_tokens_for_instance(*context*, *instance_uuid*)

target = <Target version=2.1>

3.7.345 The nova.consoleauth.rpcapi Module

Client side of the consoleauth RPC API.

class ConsoleAuthAPI
Bases: object

Client side of the consoleauth rpc API.

API version history:

- 1.0 - Initial version.

•1.1 - Added get_backdoor_port()

•1.2 - Added instance_uuid to authorize_console, and delete_tokens_for_instance

... Grizzly and Havana support message version 1.2. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 1.2.

•2.0 - Major API rev for Icehouse

... Icehouse and Juno support message version 2.0. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.0.

•2.1 - Added access_url to authorize_console

... Kilo support message version 2.1. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.1.

VERSION_ALIASES = {'kilo': '2.1', 'grizzly': '1.2', 'havana': '1.2', 'juno': '2.0', 'icehouse': '2.0'}

authorize_console(*ctxt*, *token*, *console_type*, *host*, *port*, *internal_access_path*, *instance_uuid*, *access_url*)

check_token(*ctxt*, *token*)

delete_tokens_for_instance(*ctxt*, *instance_uuid*)

3.7.346 The nova.context Module

RequestContext: context for requests that persist through all of nova.

class RequestContext(*user_id=None*, *project_id=None*, *is_admin=None*, *read_deleted='no'*, *roles=None*, *remote_address=None*, *timestamp=None*, *request_id=None*, *auth_token=None*, *overwrite=True*, *quota_class=None*, *user_name=None*, *project_name=None*, *service_catalog=None*, *instance_lock_checked=False*, *user_auth_plugin=None*, ***kwargs*)

Bases: oslo_context.context.RequestContext

Security context and request information.

Represents the user taking a given action within the system.

elevated(*read_deleted=None*, *overwrite=False*)

Return a version of this context with admin flag set.

classmethod from_dict(*values*)

get_auth_plugin()

read_deleted

to_dict()

authorize_project_context(*context*, *project_id*)

Ensures a request has permission to access the given project.

authorize_quota_class_context(*context*, *class_name*)

Ensures a request has permission to access the given quota class.

authorize_user_context(*context*, *user_id*)

Ensures a request has permission to access the given user.

get_admin_context(*read_deleted='no'*)

is_user_context(*context*)

Indicates if the request context is a normal user.

require_admin_context(*ctxt*)
Raise exception.AdminRequired() if context is an admin context.

require_context(*ctxt*)
Raise exception.Forbidden() if context is not a user or an admin context.

3.7.347 The nova.crypto Module

Wrappers around standard crypto data elements.

Includes root and intermediate CAs, SSH key_pairs and x509 certificates.

ca_folder(*project_id=None*)

ca_path(*project_id=None*)

convert_from_sshrsa_to_pkcs8(*pubkey*)
Convert a ssh public key to openssl format Equivalent to the ssh-keygen's -m option

crl_path(*project_id=None*)

decrypt_text(*project_id*, *text*)

ensure_ca_filesystem()
Ensure the CA filesystem exists.

fetch_ca(*project_id=None*)

fetch_crl(*project_id*)
Get crl file for project.

generate_fingerprint(*public_key*)

generate_key_pair(*bits=2048*)

generate_vpn_files(*project_id*)

generate_winrm_x509_cert(*user_id*, *bits=2048*)
Generate a cert for passwordless auth for user in project.

generate_x509_cert(*user_id*, *project_id*, *bits=2048*)
Generate and sign a cert for user in project.

generate_x509_fingerprint(*pem_key*)

key_path(*project_id=None*)

revoke_cert(*project_id*, *file_name*)
Revoke a cert by file name.

revoke_certs_by_project(*project_id*)
Revoke all project certs.

revoke_certs_by_user(*user_id*)
Revoke all user certs.

revoke_certs_by_user_and_project(*user_id*, *project_id*)
Revoke certs for user in project.

sign_csr(*csr_text*, *project_id=None*)

ssh_encrypt_text(*ssh_public_key*, *text*)
Encrypt text with an ssh public key.

3.7.348 The nova.db.api Module

Defines interface for DB access.

Functions in this module are imported into the nova.db namespace. Call these functions from nova.db namespace, not the nova.db.api namespace.

All functions in this module return objects that implement a dictionary-like interface. Currently, many of these objects are sqlalchemy objects that implement a dictionary interface. However, a future goal is to have all of these objects be simple dictionaries.

action_event_finish(*context*, *values*)
Finish an event on an instance action.

action_event_get_by_id(*context*, *action_id*, *event_id*)

action_event_start(*context*, *values*)
Start an event on an instance action.

action_events_get(*context*, *action_id*)
Get the events by action id.

action_finish(*context*, *values*)
Finish an action for an instance.

action_get_by_request_id(*context*, *uuid*, *request_id*)
Get the action by request_id and given instance.

action_start(*context*, *values*)
Start an action for an instance.

actions_get(*context*, *uuid*)
Get all instance actions for the provided instance.

agent_build_create(*context*, *values*)
Create a new agent build entry.

agent_build_destroy(*context*, *agent_update_id*)
Destroy agent build entry.

agent_build_get_all(*context*, *hypervisor=None*)
Get all agent builds.

agent_build_get_by_triple(*context*, *hypervisor*, *os*, *architecture*)
Get agent build by hypervisor/OS/architecture triple.

agent_build_update(*context*, *agent_build_id*, *values*)
Update agent build entry.

aggregate_create(*context*, *values*, *metadata=None*)
Create a new aggregate with metadata.

aggregate_delete(*context*, *aggregate_id*)
Delete an aggregate.

aggregate_get(*context*, *aggregate_id*)
Get a specific aggregate by id.

aggregate_get_all(*context*)
Get all aggregates.

aggregate_get_by_host(*context*, *host*, *key=None*)
Get a list of aggregates that host belongs to.

aggregate_get_by_metadata_key(*context*, *key*)

aggregate_host_add(*context*, *aggregate_id*, *host*)
Add host to the aggregate.

aggregate_host_delete(*context*, *aggregate_id*, *host*)
Delete the given host from the aggregate.

aggregate_host_get_all(*context*, *aggregate_id*)
Get hosts for the specified aggregate.

aggregate_metadata_add(*context*, *aggregate_id*, *metadata*, *set_delete=False*)
Add/update metadata. If set_delete=True, it adds only.

aggregate_metadata_delete(*context*, *aggregate_id*, *key*)
Delete the given metadata key.

aggregate_metadata_get(*context*, *aggregate_id*)
Get metadata for the specified aggregate.

aggregate_metadata_get_by_host(*context*, *host*, *key=None*)
Get metadata for all aggregates that host belongs to.

Returns a dictionary where each value is a set, this is to cover the case where there two aggregates have different values for the same key. Optional key filter

aggregate_update(*context*, *aggregate_id*, *values*)
Update the attributes of an aggregates.

If values contains a metadata key, it updates the aggregate metadata too.

archive_deleted_rows(*context*, *max_rows=None*)
Move up to max_rows rows from production tables to corresponding shadow tables.

Returns number of rows archived.

archive_deleted_rows_for_table(*context*, *tablename*, *max_rows=None*)
Move up to max_rows rows from tablename to corresponding shadow table.

Returns number of rows archived.

block_device_mapping_create(*context*, *values*, *legacy=True*)
Create an entry of block device mapping.

block_device_mapping_destroy(*context*, *bdm_id*)
Destroy the block device mapping.

block_device_mapping_destroy_by_instance_and_device(*context*, *instance_uuid*, *device_name*)
Destroy the block device mapping.

block_device_mapping_destroy_by_instance_and_volume(*context*, *instance_uuid*, *volume_id*)
Destroy the block device mapping.

block_device_mapping_get_all_by_instance(*context*, *instance_uuid*, *use_slave=False*)
Get all block device mapping belonging to an instance.

block_device_mapping_get_by_volume_id(*context*, *volume_id*, *columns_to_join=None*)
Get block device mapping for a given volume.

block_device_mapping_update(*context*, *bdm_id*, *values*, *legacy=True*)
Update an entry of block device mapping.

block_device_mapping_update_or_create(*context*, *values*, *legacy=True*)
Update an entry of block device mapping.

If not existed, create a new entry

bw_usage_get(*context*, *uuid*, *start_period*, *mac*, *use_slave=False*)
Return bw usage for instance and mac in a given audit period.

bw_usage_get_by_uuids(*context*, *uuids*, *start_period*, *use_slave=False*)
Return bw usages for instance(s) in a given audit period.

bw_usage_update(*context*, *uuid*, *mac*, *start_period*, *bw_in*, *bw_out*, *last_ctr_in*, *last_ctr_out*, *last_refreshed=None*, *update_cells=True*)
Update cached bandwidth usage for an instance's network based on mac address. Creates new record if needed.

cell_create(*context*, *values*)
Create a new child Cell entry.

cell_delete(*context*, *cell_name*)
Delete a child Cell.

cell_get(*context*, *cell_name*)
Get a specific child Cell.

cell_get_all(*context*)
Get all child Cells.

cell_update(*context*, *cell_name*, *values*)
Update a child Cell entry.

certificate_create(*context*, *values*)
Create a certificate from the values dictionary.

certificate_get_all_by_project(*context*, *project_id*)
Get all certificates for a project.

certificate_get_all_by_user(*context*, *user_id*)
Get all certificates for a user.

certificate_get_all_by_user_and_project(*context*, *user_id*, *project_id*)
Get all certificates for a user and project.

compute_node_create(*context*, *values*)
Create a compute node from the values dictionary.

Parameters

- **context** – The security context
- **values** – Dictionary containing compute node properties

Returns Dictionary-like object containing the properties of the created node, including its corresponding service and statistics

compute_node_delete(*context*, *compute_id*)
Delete a compute node from the database.

Parameters

- **context** – The security context
- **compute_id** – ID of the compute node

Raises ComputeHostNotFound if compute node with the given ID doesn't exist.

compute_node_get(*context*, *compute_id*)
Get a compute node by its id.

Parameters

- **context** – The security context
- **compute_id** – ID of the compute node

Returns Dictionary-like object containing properties of the compute node

Raises ComputeHostNotFound if compute node with the given ID doesn't exist.

compute_node_get_all(*context*)
Get all computeNodes.

Parameters **context** – The security context

Returns List of dictionaries each containing compute node properties

compute_node_get_all_by_host(*context*, *host*, *use_slave=False*)
Get compute nodes by host name

Parameters

- **context** – The security context (admin)
- **host** – Name of the host

Returns List of dictionaries each containing compute node properties

compute_node_get_by_host_and_nodename(*context*, *host*, *nodename*)
Get a compute node by its associated host and nodename.

Parameters

- **context** – The security context (admin)
- **host** – Name of the host
- **nodename** – Name of the node

Returns Dictionary-like object containing properties of the compute node, including its statistics

Raises ComputeHostNotFound if host with the given name doesn't exist.

compute_node_search_by_hypervisor(*context*, *hypervisor_match*)
Get compute nodes by hypervisor hostname.

Parameters

- **context** – The security context
- **hypervisor_match** – The hypervisor hostname

Returns List of dictionary-like objects each containing compute node properties

compute_node_statistics(*context*)
Get aggregate statistics over all compute nodes.

Parameters **context** – The security context

Returns Dictionary containing compute node characteristics summed up over all the compute nodes, e.g. 'vcpus', 'free_ram_mb' etc.

compute_node_update(*context*, *compute_id*, *values*)
Set the given properties on a compute node and update it.

Parameters

- **context** – The security context
- **compute_id** – ID of the compute node
- **values** – Dictionary containing compute node properties to be updated

Returns Dictionary-like object containing the properties of the updated compute node, including its corresponding service and statistics

Raises ComputeHostNotFound if compute node with the given ID doesn't exist.

compute_nodes_get_by_service_id(*context*, *service_id*)
Get a list of compute nodes by their associated service id.

Parameters

- **context** – The security context
- **service_id** – ID of the associated service

Returns List of dictionary-like objects, each containing properties of the compute node, including its corresponding service and statistics

Raises ServiceNotFound if service with the given ID doesn't exist.

console_create(*context*, *values*)
Create a console.

console_delete(*context*, *console_id*)
Delete a console.

console_get(*context*, *console_id*, *instance_uuid=None*)
Get a specific console (possibly on a given instance).

console_get_all_by_instance(*context*, *instance_uuid*, *columns_to_join=None*)
Get consoles for a given instance.

console_get_by_pool_instance(*context*, *pool_id*, *instance_uuid*)
Get console entry for a given instance and pool.

console_pool_create(*context*, *values*)
Create console pool.

console_pool_get_all_by_host_type(*context*, *host*, *console_type*)
Fetch all pools for given proxy host and type.

console_pool_get_by_host_type(*context*, *compute_host*, *proxy_host*, *console_type*)
Fetch a console pool for a given proxy host, compute host, and type.

constraint(***conditions*)
Return a constraint object suitable for use with some updates.

dnsdomain_get(*context*, *fqdomain*)
Get the db record for the specified domain.

dnsdomain_get_all(*context*)
Get a list of all dnsdomains in our database.

dnsdomain_register_for_project(*context*, *fqdomain*, *project*)
Associated a DNS domain with a project id.

dnsdomain_register_for_zone(*context*, *fqdomain*, *zone*)
Associated a DNS domain with an availability zone.

dnsdomain_unregister(*context*, *fqdomain*)
Purge associations for the specified DNS zone.

ec2_instance_create(*context*, *instance_uuid*, *id=None*)
Create the ec2 id to instance uuid mapping on demand.

ec2_instance_get_by_id(*context*, *instance_id*)

ec2_instance_get_by_uuid(*context*, *instance_uuid*)

ec2_snapshot_create(*context*, *snapshot_id*, *forced_id=None*)

ec2_snapshot_get_by_ec2_id(*context*, *ec2_id*)

ec2_snapshot_get_by_uuid(*context*, *snapshot_uuid*)

ec2_volume_create(*context*, *volume_id*, *forced_id=None*)

ec2_volume_get_by_id(*context*, *volume_id*)

ec2_volume_get_by_uuid(*context*, *volume_uuid*)

equal_any(**values*)
Return an equality condition object suitable for use in a constraint.

Equal_any conditions require that a model object's attribute equal any one of the given values.

fixed_ip_associate(*context*, *address*, *instance_uuid*, *network_id=None*, *reserved=False*)
Associate fixed ip to instance.

Raises if fixed ip is not available.

fixed_ip_associate_pool(*context*, *network_id*, *instance_uuid=None*, *host=None*)
Find free ip in network and associate it to instance or host.

Raises if one is not available.

fixed_ip_bulk_create(*context*, *ips*)
Create a lot of fixed ips from the values dictionary.

fixed_ip_create(*context*, *values*)
Create a fixed ip from the values dictionary.

fixed_ip_disassociate(*context*, *address*)
Disassociate a fixed ip from an instance by address.

fixed_ip_disassociate_all_by_timeout(*context*, *host*, *time*)
Disassociate old fixed ips from host.

fixed_ip_get(*context*, *id*, *get_network=False*)
Get fixed ip by id or raise if it does not exist.

If get_network is true, also return the associated network.

fixed_ip_get_all(*context*)
Get all defined fixed ips.

fixed_ip_get_by_address(*context*, *address*, *columns_to_join=None*)
Get a fixed ip by address or raise if it does not exist.

fixed_ip_get_by_floating_address(*context*, *floating_address*)
Get a fixed ip by a floating address.

fixed_ip_get_by_host(*context*, *host*)
Get fixed ips by compute host.

fixed_ip_get_by_instance(*context*, *instance_uuid*)
Get fixed ips by instance or raise if none exist.

fixed_ip_get_by_network_host(*context*, *network_uuid*, *host*)
Get fixed ip for a host in a network.

fixed_ip_update(*context*, *address*, *values*)
Create a fixed ip from the values dictionary.

fixed_ips_by_virtual_interface(*context*, *vif_id*)
Get fixed ips by virtual interface or raise if none exist.

flavor_access_add(*context*, *flavor_id*, *project_id*)
Add flavor access for project.

flavor_access_get_by_flavor_id(*context*, *flavor_id*)
Get flavor access by flavor id.

flavor_access_remove(*context*, *flavor_id*, *project_id*)
Remove flavor access for project.

flavor_create(*context*, *values*, *projects=None*)
Create a new instance type.

flavor_destroy(*context*, *name*)
Delete an instance type.

flavor_extra_specs_delete(*context*, *flavor_id*, *key*)
Delete the given extra specs item.

flavor_extra_specs_get(*context*, *flavor_id*)
Get all extra specs for an instance type.

flavor_extra_specs_update_or_create(*context*, *flavor_id*, *extra_specs*)
Create or update instance type extra specs.

This adds or modifies the key/value pairs specified in the extra specs dict argument

flavor_get(*context*, *id*)
Get instance type by id.

flavor_get_all(*context*, *inactive=False*, *filters=None*, *sort_key='flavorid'*, *sort_dir='asc'*, *limit=None*, *marker=None*)
Get all instance flavors.

flavor_get_by_flavor_id(*context*, *id*, *read_deleted=None*)
Get instance type by flavor id.

flavor_get_by_name(*context*, *name*)
Get instance type by name.

floating_ip_allocate_address(*context*, *project_id*, *pool*, *auto_assigned=False*)
Allocate free floating ip from specified pool and return the address.

Raises if one is not available.

floating_ip_bulk_create(*context*, *ips*, *want_result=True*)
Create a lot of floating ips from the values dictionary. :param want_result: If set to True, return floating ips inserted

floating_ip_bulk_destroy(*context*, *ips*)
Destroy a lot of floating ips from the values dictionary.

floating_ip_create(*context*, *values*)
Create a floating ip from the values dictionary.

floating_ip_deallocate(*context*, *address*)
Deallocate a floating ip by address.

floating_ip_destroy(*context*, *address*)
Destroy the floating_ip or raise if it does not exist.

floating_ip_disassociate(*context*, *address*)
Disassociate a floating ip from a fixed ip by address.

Returns the fixed ip record joined to network record or None if the ip was not associated to an ip.

floating_ip_fixed_ip_associate(*context*, *floating_address*, *fixed_address*, *host*)
Associate a floating ip to a fixed_ip by address.

Returns the fixed ip record joined to network record or None if the ip was already associated to the fixed ip.

floating_ip_get(*context*, *id*)

floating_ip_get_all(*context*)
Get all floating ips.

floating_ip_get_all_by_host(*context*, *host*)
Get all floating ips by host.

floating_ip_get_all_by_project(*context*, *project_id*)
Get all floating ips by project.

floating_ip_get_by_address(*context*, *address*)
Get a floating ip by address or raise if it doesn't exist.

floating_ip_get_by_fixed_address(*context*, *fixed_address*)
Get a floating ips by fixed address.

floating_ip_get_by_fixed_ip_id(*context*, *fixed_ip_id*)
Get a floating ips by fixed address.

floating_ip_get_pools(*context*)
Returns a list of floating ip pools.

floating_ip_update(*context*, *address*, *values*)
Update a floating ip by address or raise if it doesn't exist.

get_instance_uuid_by_ec2_id(*context*, *ec2_id*)
Get uuid through ec2 id from instance_id_mappings table.

instance_add_security_group(*context*, *instance_id*, *security_group_id*)
Associate the given security group with the given instance.

instance_create(*context*, *values*)
Create an instance from the values dictionary.

instance_destroy(*context*, *instance_uuid*, *constraint=None*)
Destroy the instance or raise if it does not exist.

instance_extra_get_by_instance_uuid(*context*, *instance_uuid*, *columns=None*)
Get the instance extra record

Parameters

- **instance_uuid** – = uuid of the instance tied to the topology record
- **columns** – A list of the columns to load, or None for 'all of them'

instance_extra_update_by_uuid(*context*, *instance_uuid*, *updates*)
Update the instance extra record by instance uuid

Parameters

- **instance_uuid** – = uuid of the instance tied to the record
- **updates** – A dict of updates to apply

instance_fault_create(*context*, *values*)
Create a new Instance Fault.

instance_fault_get_by_instance_uuids(*context*, *instance_uuids*)
Get all instance faults for the provided instance_uuids.

instance_floating_address_get_all(*context*, *instance_uuid*)
Get all floating ip addresses of an instance.

instance_get(*context*, *instance_id*, *columns_to_join=None*)
Get an instance or raise if it does not exist.

instance_get_active_by_window_joined(*context*, *begin*, *end=None*, *project_id=None*, *host=None*, *use_slave=False*, *columns_to_join=None*)
Get instances and joins active during a certain time window.

Specifying a project_id will filter for a certain project. Specifying a host will filter for instances on a given compute host.

instance_get_all(*context*, *columns_to_join=None*)
Get all instances.

instance_get_all_by_filters(*context*, *filters*, *sort_key='created_at'*, *sort_dir='desc'*, *limit=None*, *marker=None*, *columns_to_join=None*, *use_slave=False*)
Get all instances that match all filters.

instance_get_all_by_filters_sort(*context*, *filters*, *limit=None*, *marker=None*, *columns_to_join=None*, *use_slave=False*, *sort_keys=None*, *sort_dirs=None*)
Get all instances that match all filters sorted by multiple keys.

sort_keys and sort_dirs must be a list of strings.

instance_get_all_by_host(*context*, *host*, *columns_to_join=None*, *use_slave=False*)
Get all instances belonging to a host.

instance_get_all_by_host_and_node(*context*, *host*, *node*, *columns_to_join=None*)
Get all instances belonging to a node.

instance_get_all_by_host_and_not_type(*context*, *host*, *type_id=None*)
Get all instances belonging to a host with a different type_id.

instance_get_by_uuid(*context*, *uuid*, *columns_to_join=None*, *use_slave=False*)
Get an instance or raise if it does not exist.

instance_group_create(*context*, *values*, *policies=None*, *members=None*)
Create a new group.

Each group will receive a unique uuid. This will be used for access to the group.

instance_group_delete(*context*, *group_uuid*)
Delete an group.

instance_group_get(*context*, *group_uuid*)
Get a specific group by id.

instance_group_get_all(*context*)
Get all groups.

instance_group_get_all_by_project_id(*context*, *project_id*)
Get all groups for a specific project_id.

instance_group_get_by_instance(*context*, *instance_uuid*)
Get the group an instance is a member of.

instance_group_member_delete(*context*, *group_uuid*, *instance_id*)
Delete a specific member from the group.

instance_group_members_add(*context*, *group_uuid*, *members*, *set_delete=False*)
Add members to the group.

instance_group_members_get(*context*, *group_uuid*)
Get the members from the group.

instance_group_update(*context*, *group_uuid*, *values*)
Update the attributes of an group.

instance_info_cache_delete(*context*, *instance_uuid*)
Deletes an existing instance_info_cache record

Parameters **instance_uuid** – = uuid of the instance tied to the cache record

instance_info_cache_get(*context*, *instance_uuid*)
Gets an instance info cache from the table.

Parameters **instance_uuid** – = uuid of the info cache's instance

instance_info_cache_update(*context*, *instance_uuid*, *values*)
Update an instance info cache record in the table.

Parameters

- **instance_uuid** – = uuid of info cache's instance
- **values** – = dict containing column values to update

instance_metadata_delete(*context*, *instance_uuid*, *key*)
Delete the given metadata item.

instance_metadata_get(*context*, *instance_uuid*)
Get all metadata for an instance.

instance_metadata_update(*context*, *instance_uuid*, *metadata*, *delete*)
Update metadata if it exists, otherwise create it.

instance_remove_security_group(*context*, *instance_id*, *security_group_id*)
Disassociate the given security group from the given instance.

instance_system_metadata_get(*context*, *instance_uuid*)
Get all system metadata for an instance.

instance_system_metadata_update(*context*, *instance_uuid*, *metadata*, *delete*)
Update metadata if it exists, otherwise create it.

instance_tag_add(*context*, *instance_uuid*, *tag*)
Add tag to the instance.

instance_tag_delete(*context*, *instance_uuid*, *tag*)
Delete specified tag from the instance.

instance_tag_delete_all(*context*, *instance_uuid*)
Delete all tags from the instance.

instance_tag_exists(*context*, *instance_uuid*, *tag*)
Check if specified tag exist on the instance.

instance_tag_get_by_instance_uuid(*context*, *instance_uuid*)
Get all tags for a given instance.

instance_tag_set(*context*, *instance_uuid*, *tags*)
Replace all of the instance tags with specified list of tags.

instance_update(*context*, *instance_uuid*, *values*)
Set the given properties on an instance and update it.

Raises NotFound if instance does not exist.

instance_update_and_get_original(*context*, *instance_uuid*, *values*, *columns_to_join=None*)
Set the given properties on an instance and update it. Return a shallow copy of the original instance reference, as well as the updated one.

Parameters

- **context** – = request context object
- **instance_uuid** – = instance id or uuid
- **values** – = dict containing column values

Returns a tuple of the form (old_instance_ref, new_instance_ref)

Raises NotFound if instance does not exist.

key_pair_count_by_user(*context*, *user_id*)
Count number of key pairs for the given user ID.

key_pair_create(*context*, *values*)
Create a key_pair from the values dictionary.

key_pair_destroy(*context*, *user_id*, *name*)
Destroy the key_pair or raise if it does not exist.

key_pair_get(*context*, *user_id*, *name*)
Get a key_pair or raise if it does not exist.

key_pair_get_all_by_user(*context*, *user_id*)
Get all key_pairs by user.

migration_create(*context*, *values*)
Create a migration record.

migration_get(*context*, *migration_id*)
Finds a migration by the id.

migration_get_all_by_filters(*context*, *filters*)
Finds all migrations in progress.

migration_get_by_instance_and_status(*context*, *instance_uuid*, *status*)
Finds a migration by the instance uuid its migrating.

migration_get_in_progress_by_host_and_node(*context*, *host*, *node*)
Finds all migrations for the given host + node that are not yet confirmed or reverted.

migration_get_unconfirmed_by_dest_compute(*context*, *confirm_window*, *dest_compute*, *use_slave=False*)
Finds all unconfirmed migrations within the confirmation window for a specific destination compute host.

migration_update(*context*, *id*, *values*)
Update a migration instance.

network_associate(*context*, *project_id*, *network_id=None*, *force=False*)
Associate a free network to a project.

network_count_reserved_ips(*context*, *network_id*)
Return the number of reserved ips in the network.

network_create_safe(*context*, *values*)
Create a network from the values dict.

The network is only returned if the create succeeds. If the create violates constraints because the network already exists, no exception is raised.

network_delete_safe(*context*, *network_id*)
Delete network with key network_id.

This method assumes that the network is not associated with any project

network_disassociate(*context*, *network_id*, *disassociate_host=True*, *disassociate_project=True*)
Disassociate the network from project or host

Raises if it does not exist.

network_get(*context*, *network_id*, *project_only='allow_none'*)
Get a network or raise if it does not exist.

network_get_all(*context*, *project_only='allow_none'*)
Return all defined networks.

network_get_all_by_host(*context*, *host*)
All networks for which the given host is the network host.

network_get_all_by_uuids(*context*, *network_uuids*, *project_only='allow_none'*)
Return networks by ids.

network_get_associated_fixed_ips(*context*, *network_id*, *host=None*)
Get all network's ips that have been associated.

network_get_by_cidr(*context*, *cidr*)
Get a network by cidr or raise if it does not exist.

network_get_by_uuid(*context*, *uuid*)
Get a network by uuid or raise if it does not exist.

network_in_use_on_host(*context*, *network_id*, *host=None*)
Indicates if a network is currently in use on host.

network_set_host(*context*, *network_id*, *host_id*)
Safely set the host for network.

network_update(*context*, *network_id*, *values*)
Set the given properties on a network and update it.

Raises NotFound if network does not exist.

not_equal(**values*)
Return an inequality condition object suitable for use in a constraint.

Not_equal conditions require that a model object's attribute differs from all of the given values.

pci_device_destroy(*context*, *node_id*, *address*)
Delete a PCI device record.

pci_device_get_all_by_instance_uuid(*context*, *instance_uuid*)
Get PCI devices allocated to instance.

pci_device_get_all_by_node(*context*, *node_id*)
Get all PCI devices for one host.

pci_device_get_by_addr(*context*, *node_id*, *dev_addr*)
Get PCI device by address.

pci_device_get_by_id(*context*, *id*)
Get PCI device by id.

pci_device_update(*context*, *node_id*, *address*, *value*)
Update a pci device.

project_get_networks(*context*, *project_id*, *associate=True*)
Return the network associated with the project.

If associate is true, it will attempt to associate a new network if one is not found, otherwise it returns None.

provider_fw_rule_create(*context*, *rule*)
Add a firewall rule at the provider level (all hosts & instances).

provider_fw_rule_destroy(*context*, *rule_id*)
Delete a provider firewall rule from the database.

provider_fw_rule_get_all(*context*)
Get all provider-level firewall rules.

quota_class_create(*context*, *class_name*, *resource*, *limit*)
Create a quota class for the given name and resource.

quota_class_get(*context*, *class_name*, *resource*)
Retrieve a quota class or raise if it does not exist.

quota_class_get_all_by_name(*context*, *class_name*)
Retrieve all quotas associated with a given quota class.

quota_class_get_default(*context*)
Retrieve all default quotas.

quota_class_update(*context*, *class_name*, *resource*, *limit*)
Update a quota class or raise if it does not exist.

quota_create(*context*, *project_id*, *resource*, *limit*, *user_id=None*)
Create a quota for the given project and resource.

quota_destroy_all_by_project(*context*, *project_id*)
Destroy all quotas associated with a given project.

quota_destroy_all_by_project_and_user(*context*, *project_id*, *user_id*)
Destroy all quotas associated with a given project and user.

quota_get(*context*, *project_id*, *resource*, *user_id=None*)
Retrieve a quota or raise if it does not exist.

quota_get_all(*context*, *project_id*)
Retrieve all user quotas associated with a given project.

quota_get_all_by_project(*context*, *project_id*)
Retrieve all quotas associated with a given project.

quota_get_all_by_project_and_user(*context*, *project_id*, *user_id*)
Retrieve all quotas associated with a given project and user.

quota_reserve(*context*, *resources*, *quotas*, *user_quotas*, *deltas*, *expire*, *until_refresh*, *max_age*, *project_id=None*, *user_id=None*)
Check quotas and create appropriate reservations.

quota_update(*context*, *project_id*, *resource*, *limit*, *user_id=None*)
Update a quota or raise if it does not exist.

quota_usage_get(*context*, *project_id*, *resource*, *user_id=None*)
Retrieve a quota usage or raise if it does not exist.

quota_usage_get_all_by_project(*context*, *project_id*)
Retrieve all usage associated with a given resource.

quota_usage_get_all_by_project_and_user(*context*, *project_id*, *user_id*)
Retrieve all usage associated with a given resource.

quota_usage_update(*context*, *project_id*, *user_id*, *resource*, ***kwargs*)
Update a quota usage or raise if it does not exist.

reservation_commit(*context*, *reservations*, *project_id=None*, *user_id=None*)
Commit quota reservations.

reservation_expire(*context*)
Roll back any expired reservations.

reservation_rollback(*context*, *reservations*, *project_id=None*, *user_id=None*)
Roll back quota reservations.

s3_image_create(*context*, *image_uuid*)
Create local s3 image represented by provided uuid.

s3_image_get(*context*, *image_id*)
Find local s3 image represented by the provided id.

s3_image_get_by_uuid(*context*, *image_uuid*)
Find local s3 image represented by the provided uuid.

security_group_create(*context*, *values*)
Create a new security group.

security_group_default_rule_create(*context*, *values*)

security_group_default_rule_destroy(*context*, *security_group_rule_default_id*)

security_group_default_rule_get(*context*, *security_group_rule_default_id*)

security_group_default_rule_list(*context*)

security_group_destroy(*context*, *security_group_id*)
Deletes a security group.

security_group_ensure_default(*context*)
Ensure default security group exists for a project_id.

Returns a tuple with the first element being a bool indicating if the default security group previously existed. Second element is the dict used to create the default security group.

security_group_get(*context*, *security_group_id*, *columns_to_join=None*)
Get security group by its id.

security_group_get_all(*context*)
Get all security groups.

security_group_get_by_instance(*context*, *instance_uuid*)
Get security groups to which the instance is assigned.

security_group_get_by_name(*context*, *project_id*, *group_name*, *columns_to_join=None*)
Returns a security group with the specified name from a project.

security_group_get_by_project(*context*, *project_id*)
Get all security groups belonging to a project.

security_group_in_use(*context*, *group_id*)
Indicates if a security group is currently in use.

security_group_rule_count_by_group(*context*, *security_group_id*)
Count rules in a given security group.

security_group_rule_create(*context*, *values*)
Create a new security group.

security_group_rule_destroy(*context*, *security_group_rule_id*)
Deletes a security group rule.

security_group_rule_get(*context*, *security_group_rule_id*)
Gets a security group rule.

security_group_rule_get_by_security_group(*context*, *security_group_id*, *columns_to_join=None*)
Get all rules for a given security group.

security_group_rule_get_by_security_group_grantee(*context*, *security_group_id*)
Get all rules that grant access to the given security group.

security_group_update(*context*, *security_group_id*, *values*, *columns_to_join=None*)
Update a security group.

service_create(*context*, *values*)
Create a service from the values dictionary.

service_destroy(*context*, *service_id*)
Destroy the service or raise if it does not exist.

service_get(*context*, *service_id*, *use_slave=False*)
Get a service or raise if it does not exist.

service_get_all(*context*, *disabled=None*)
Get all services.

service_get_all_by_binary(*context*, *binary*)
Get all services for a given binary.

service_get_all_by_host(*context*, *host*)
Get all services for a given host.

service_get_all_by_topic(*context*, *topic*)
Get all services for a given topic.

service_get_by_compute_host(*context*, *host*, *use_slave=False*)
Get the service entry for a given compute host.

Returns the service entry joined with the compute_node entry.

service_get_by_host_and_binary(*context*, *host*, *binary*)
Get a service by hostname and binary.

service_get_by_host_and_topic(*context*, *host*, *topic*)
Get a service by hostname and topic it listens to.

service_update(*context*, *service_id*, *values*)
Set the given properties on a service and update it.

Raises NotFound if service does not exist.

task_log_begin_task(*context*, *task_name*, *period_beginning*, *period_ending*, *host*, *task_items=None*, *message=None*)
Mark a task as started for a given host/time period.

task_log_end_task(*context*, *task_name*, *period_beginning*, *period_ending*, *host*, *errors*, *message=None*)
Mark a task as complete for a given host/time period.

task_log_get(*context*, *task_name*, *period_beginning*, *period_ending*, *host*, *state=None*)

task_log_get_all(*context*, *task_name*, *period_beginning*, *period_ending*, *host=None*, *state=None*)

virtual_interface_create(*context*, *values*)
Create a virtual interface record in the database.

virtual_interface_delete_by_instance(*context*, *instance_id*)
Delete virtual interface records associated with instance.

virtual_interface_get(*context*, *vif_id*)
Gets a virtual interface from the table.

virtual_interface_get_all(*context*)
Gets all virtual interfaces from the table.

virtual_interface_get_by_address(*context*, *address*)
Gets a virtual interface from the table filtering on address.

virtual_interface_get_by_instance(*context*, *instance_id*, *use_slave=False*)
Gets all virtual_interfaces for instance.

virtual_interface_get_by_instance_and_network(*context*, *instance_id*, *network_id*)
Gets all virtual interfaces for instance.

virtual_interface_get_by_uuid(*context*, *vif_uuid*)
Gets a virtual interface from the table filtering on vif uuid.

vol_get_usage_by_time(*context*, *begin*)
Return volumes usage that have been updated after a specified time.

vol_usage_update(*context*, *id*, *rd_req*, *rd_bytes*, *wr_req*, *wr_bytes*, *instance_id*, *project_id*, *user_id*, *availability_zone*, *update_totals=False*)
Update cached volume usage for a volume

Creates new record if needed.

3.7.349 The nova.db.base Module

Base class for classes that need modular database access.

class Base(*db_driver=None*)
Bases: `object`

DB driver is injected in the init method.

3.7.350 The nova.db.migration Module

Database setup and migration commands.

db_contract(*dryrun=False*, *database='main'*)
Contract database schema.

db_expand(*dryrun=False*, *database='main'*)
Expand database schema.

db_initial_version(*database='main'*)
The starting version for the database.

db_migrate (*dryrun=False*, *database='main'*)
Migrate database schema.

db_null_instance_uuid_scan (*delete=False*)
Utility for scanning the database to look for NULL instance uuid rows.

Scans the backing nova database to look for table entries where instances.uuid or instance_uuid columns are NULL (except for the fixed_ips table since that can contain NULL instance_uuid entries by design). Dumps the tables that have NULL instance_uuid entries or optionally deletes them based on usage.

This tool is meant to be used in conjunction with the 267 database migration script to detect and optionally cleanup NULL instance_uuid records.

Parameters **delete** – If true, delete NULL instance_uuid records found, else just query to see if they exist for reporting.

Returns dict of table name to number of hits for NULL instance_uuid rows.

db_sync (*version=None*, *database='main'*)
Migrate the database to *version* or the most recent version.

db_version (*database='main'*)
Display the current database version.

3.7.351 The nova.db.sqlalchemy.api Module

Implementation of SQLAlchemy backend.

class Constraint (*conditions*)
Bases: object

apply (*model*, *query*)

class EqualityCondition (*values*)
Bases: object

clauses (*field*)

class InequalityCondition (*values*)
Bases: object

clauses (*field*)

action_event_finish (*context*, *values*)
Finish an event on an instance action.

action_event_get_by_id (*context*, *action_id*, *event_id*)

action_event_start (*context*, *values*)
Start an event on an instance action.

action_events_get (*context*, *action_id*)

action_finish (*context*, *values*)

action_get_by_request_id (*context*, *instance_uuid*, *request_id*)
Get the action by request_id and given instance.

action_start (*context*, *values*)

actions_get (*context*, *instance_uuid*)
Get all instance actions for the provided uuid.

agent_build_create (*context*, *values*)

agent_build_destroy(*context*, *agent_build_id*)

agent_build_get_all(*context*, *hypervisor=None*)

agent_build_get_by_triple(*context*, *hypervisor*, *os*, *architecture*)

agent_build_update(*context*, *agent_build_id*, *values*)

aggregate_create(*context*, *values*, *metadata=None*)

aggregate_delete(*context*, *aggregate_id*)

aggregate_get(*context*, *aggregate_id*)

aggregate_get_all(*context*)

aggregate_get_by_host(*context*, *host*, *key=None*)
Return rows that match host (mandatory) and metadata key (optional).

:param host matches host, and is required. :param key Matches metadata key, if not None.

aggregate_get_by_metadata_key(*context*, *key*)
Return rows that match metadata key.

:param key Matches metadata key.

aggregate_host_add(*context*, *aggregate_id*, **args*, ***kwargs*)

aggregate_host_delete(*context*, *aggregate_id*, **args*, ***kwargs*)

aggregate_host_get_all(*context*, *aggregate_id*, **args*, ***kwargs*)

aggregate_metadata_add(*context*, *aggregate_id*, **args*, ***kwargs*)

aggregate_metadata_delete(*context*, *aggregate_id*, **args*, ***kwargs*)

aggregate_metadata_get(*context*, *aggregate_id*, **args*, ***kwargs*)

aggregate_metadata_get_by_host(*context*, *host*, *key=None*)

aggregate_update(*context*, *aggregate_id*, *values*)

archive_deleted_rows(**args*, ***kwargs*)
Move up to max_rows rows from production tables to the corresponding shadow tables.

Returns Number of rows archived.

archive_deleted_rows_for_table(**args*, ***kwargs*)
Move up to max_rows rows from one tables to the corresponding shadow table. The context argument is only used for the decorator.

Returns number of rows archived

block_device_mapping_create(**args*, ***kwargs*)

block_device_mapping_destroy(**args*, ***kwargs*)

block_device_mapping_destroy_by_instance_and_device(**args*, ***kwargs*)

block_device_mapping_destroy_by_instance_and_volume(**args*, ***kwargs*)

block_device_mapping_get_all_by_instance(**args*, ***kwargs*)

block_device_mapping_get_by_volume_id(**args*, ***kwargs*)

block_device_mapping_update(**args*, ***kwargs*)

block_device_mapping_update_or_create(*context*, *values*, *legacy=True*)

bw_usage_get(**args*, ***kwargs*)

bw_usage_get_by_uuids(**args*, ***kwargs*)

bw_usage_update(**args*, ***kwargs*)

cell_create(*context*, *values*)

cell_delete(*context*, *cell_name*)

cell_get(*context*, *cell_name*)

cell_get_all(*context*)

cell_update(*context*, *cell_name*, *values*)

certificate_create(**args*, ***kwargs*)

certificate_get_all_by_project(**args*, ***kwargs*)

certificate_get_all_by_user(**args*, ***kwargs*)

certificate_get_all_by_user_and_project(**args*, ***kwargs*)

compute_node_create(*context*, *values*)
Creates a new ComputeNode and populates the capacity fields with the most recent data.

compute_node_delete(*context*, *compute_id*)
Delete a ComputeNode record.

compute_node_get(*context*, *compute_id*)

compute_node_get_all(*context*)

compute_node_get_all_by_host(*context*, *host*, *use_slave=False*)

compute_node_get_by_host_and_nodename(*context*, *host*, *nodename*)

compute_node_search_by_hypervisor(*context*, *hypervisor_match*)

compute_node_statistics(*context*)
Compute statistics over all compute nodes.

compute_node_update(**args*, ***kwargs*)
Updates the ComputeNode record with the most recent data.

compute_nodes_get_by_service_id(*context*, *service_id*)

console_create(*context*, *values*)

console_delete(*context*, *console_id*)

console_get(*context*, *console_id*, *instance_uuid=None*)

console_get_all_by_instance(*context*, *instance_uuid*, *columns_to_join=None*)

console_get_by_pool_instance(*context*, *pool_id*, *instance_uuid*)

console_pool_create(*context*, *values*)

console_pool_get_all_by_host_type(*context*, *host*, *console_type*)

console_pool_get_by_host_type(*context*, *compute_host*, *host*, *console_type*)

constraint(***conditions*)

convert_objects_related_datetimes(*values*, **datetime_keys*)

dnsdomain_get(**args*, ***kwargs*)

dnsdomain_get_all(*context*)

dnsdomain_register_for_project(*context*, *fqdomain*, *project*)

dnsdomain_register_for_zone(*context*, *fqdomain*, *zone*)

dnsdomain_unregister(*context*, *fqdomain*)

ec2_instance_create(**args*, ***kwargs*)
Create ec2 compatible instance by provided uuid.

ec2_instance_get_by_id(**args*, ***kwargs*)

ec2_instance_get_by_uuid(**args*, ***kwargs*)

ec2_snapshot_create(**args*, ***kwargs*)
Create ec2 compatible snapshot by provided uuid.

ec2_snapshot_get_by_ec2_id(**args*, ***kwargs*)

ec2_snapshot_get_by_uuid(**args*, ***kwargs*)

ec2_volume_create(**args*, ***kwargs*)
Create ec2 compatible volume by provided uuid.

ec2_volume_get_by_id(**args*, ***kwargs*)

ec2_volume_get_by_uuid(**args*, ***kwargs*)

equal_any(**values*)

fixed_ip_associate(**args*, ***kwargs*)
Keyword arguments: reserved – should be a boolean value(True or False), exact value will be used to filter on the fixed ip address

fixed_ip_associate_pool(**args*, ***kwargs*)

fixed_ip_bulk_create(**args*, ***kwargs*)

fixed_ip_create(**args*, ***kwargs*)

fixed_ip_disassociate(**args*, ***kwargs*)

fixed_ip_disassociate_all_by_timeout(*context*, *host*, *time*)

fixed_ip_get(**args*, ***kwargs*)

fixed_ip_get_all(*context*)

fixed_ip_get_by_address(**args*, ***kwargs*)

fixed_ip_get_by_floating_address(**args*, ***kwargs*)

fixed_ip_get_by_host(*context*, *host*)

fixed_ip_get_by_instance(**args*, ***kwargs*)

fixed_ip_get_by_network_host(**args*, ***kwargs*)

fixed_ip_update(**args*, ***kwargs*)

fixed_ips_by_virtual_interface(**args*, ***kwargs*)

flavor_access_add(*context*, *flavor_id*, *project_id*)
Add given tenant to the flavor access list.

flavor_access_get_by_flavor_id(*context*, *flavor_id*)
Get flavor access list by flavor id.

flavor_access_remove(*context*, *flavor_id*, *project_id*)
Remove given tenant from the flavor access list.

flavor_create(*context*, *values*, *projects=None*)
Create a new instance type. In order to pass in extra specs, the values dict should contain a 'extra_specs' key/value pair:

{'extra_specs' : {'k1': 'v1', 'k2': 'v2', ...}}

flavor_destroy(*context*, *name*)
Marks specific flavor as deleted.

flavor_extra_specs_delete(**args*, ***kwargs*)

flavor_extra_specs_get(**args*, ***kwargs*)

flavor_extra_specs_update_or_create(**args*, ***kwargs*)

flavor_get(**args*, ***kwargs*)
Returns a dict describing specific flavor.

flavor_get_all(**args*, ***kwargs*)
Returns all flavors.

flavor_get_by_flavor_id(**args*, ***kwargs*)
Returns a dict describing specific flavor_id.

flavor_get_by_name(**args*, ***kwargs*)
Returns a dict describing specific flavor.

floating_ip_allocate_address(**args*, ***kwargs*)

floating_ip_bulk_create(**args*, ***kwargs*)

floating_ip_bulk_destroy(**args*, ***kwargs*)

floating_ip_create(**args*, ***kwargs*)

floating_ip_deallocate(**args*, ***kwargs*)

floating_ip_destroy(**args*, ***kwargs*)

floating_ip_disassociate(**args*, ***kwargs*)

floating_ip_fixed_ip_associate(**args*, ***kwargs*)

floating_ip_get(**args*, ***kwargs*)

floating_ip_get_all(*context*)

floating_ip_get_all_by_host(*context*, *host*)

floating_ip_get_all_by_project(**args*, ***kwargs*)

floating_ip_get_by_address(**args*, ***kwargs*)

floating_ip_get_by_fixed_address(**args*, ***kwargs*)

floating_ip_get_by_fixed_ip_id(**args*, ***kwargs*)

floating_ip_get_pools(**args*, ***kwargs*)

floating_ip_update(**args*, ***kwargs*)

get_api_engine()

get_api_session(***kwargs*)

get_backend()
The backend is this module itself.

get_engine(*use_slave=False*)

get_instance_uuid_by_ec2_id(**args*, ***kwargs*)

get_session(*use_slave=False*, ***kwargs*)

instance_add_security_group(*context*, *instance_uuid*, *security_group_id*)
Associate the given security group with the given instance.

instance_create(**args*, ***kwargs*)
Create a new Instance record in the database.

context - request context object values - dict containing column values.

instance_destroy(**args*, ***kwargs*)

instance_extra_get_by_instance_uuid(*context*, *instance_uuid*, *columns=None*)

instance_extra_update_by_uuid(*context*, *instance_uuid*, *values*)

instance_fault_create(*context*, *values*)
Create a new InstanceFault.

instance_fault_get_by_instance_uuids(*context*, *instance_uuids*)
Get all instance faults for the provided instance_uuids.

instance_floating_address_get_all(**args*, ***kwargs*)

instance_get(**args*, ***kwargs*)

instance_get_active_by_window_joined(**args*, ***kwargs*)
Return instances and joins that were active during window.

instance_get_all(**args*, ***kwargs*)

instance_get_all_by_filters(**args*, ***kwargs*)
Return instances matching all filters sorted by the primary key.

See instance_get_all_by_filters_sort for more information.

instance_get_all_by_filters_sort(**args*, ***kwargs*)
Return instances that match all filters sorted the the given keys. Deleted instances will be returned by default, unless there's a filter that says otherwise.

Depending on the name of a filter, matching for that filter is performed using either exact matching or as regular expression matching. Exact matching is applied for the following filters:

```
|   ['project_id', 'user_id', 'image_ref',
|    'vm_state', 'instance_type_id', 'uuid',
|    'metadata', 'host', 'system_metadata']
```

A third type of filter (also using exact matching), filters based on instance metadata tags when supplied under a special key named 'filter':

```
|   filters = {
|       'filter': [
|           {'name': 'tag-key', 'value': '<metakey>'},
|           {'name': 'tag-value', 'value': '<metaval>'},
|           {'name': 'tag:<metakey>', 'value': '<metaval>'}
|       ]
|   }
```

Special keys are used to tweek the query further:

```
|   'changes-since' - only return instances updated after
|   'deleted' - only return (or exclude) deleted instances
|   'soft_deleted' - modify behavior of 'deleted' to either
|                    include or exclude instances whose
|                    vm_state is SOFT_DELETED.
```

A fourth type of filter (also using exact matching), filters based on instance tags (not metadata tags). There are two types of these tags:

***tag* – One or more strings that will be used to filter results** in an AND expression.

***tag-any* – One or more strings that will be used to filter results in** an OR expression.

Tags should be represented as list:

```
|    filters = {
|        'tag': [some-tag, some-another-tag],
|        'tag-any: [some-any-tag, some-another-any-tag]
|    }
```

instance_get_all_by_host(*context*, *host*, *columns_to_join=None*, *use_slave=False*)

instance_get_all_by_host_and_node(*context*, *host*, *node*, *columns_to_join=None*)

instance_get_all_by_host_and_not_type(**args*, ***kwargs*)

instance_get_by_uuid(**args*, ***kwargs*)

instance_group_create(*context*, *values*, *policies=None*, *members=None*)
Create a new group.

instance_group_delete(*context*, *group_uuid*)
Delete an group.

instance_group_get(*context*, *group_uuid*)
Get a specific group by uuid.

instance_group_get_all(*context*)
Get all groups.

instance_group_get_all_by_project_id(*context*, *project_id*)
Get all groups.

instance_group_get_by_instance(*context*, *instance_uuid*)

instance_group_member_delete(*context*, *group_uuid*, *instance_id*)

instance_group_members_add(*context*, *group_uuid*, *members*, *set_delete=False*)

instance_group_members_get(*context*, *group_uuid*)

instance_group_update(*context*, *group_uuid*, *values*)
Update the attributes of an group.

If values contains a metadata key, it updates the aggregate metadata too. Similarly for the policies and members.

instance_info_cache_delete(**args*, ***kwargs*)
Deletes an existing instance_info_cache record

Parameters **instance_uuid** – = uuid of the instance tied to the cache record

instance_info_cache_get(**args*, ***kwargs*)
Gets an instance info cache from the table.

Parameters **instance_uuid** – = uuid of the info cache's instance

instance_info_cache_update(**args*, ***kwargs*)
Update an instance info cache record in the table.

Parameters

- **instance_uuid** – = uuid of info cache's instance
- **values** – = dict containing column values to update

instance_metadata_delete(**args*, ***kwargs*)

instance_metadata_get(**args*, ***kwargs*)

instance_metadata_update(**args*, ***kwargs*)

instance_remove_security_group(**args*, ***kwargs*)
Disassociate the given security group from the given instance.

instance_system_metadata_get(**args*, ***kwargs*)

instance_system_metadata_update(**args*, ***kwargs*)

instance_tag_add(*context*, *instance_uuid*, *tag*)

instance_tag_delete(*context*, *instance_uuid*, *tag*)

instance_tag_delete_all(*context*, *instance_uuid*)

instance_tag_exists(*context*, *instance_uuid*, *tag*)

instance_tag_get_by_instance_uuid(*context*, *instance_uuid*)

instance_tag_set(*context*, *instance_uuid*, *tags*)

instance_update(**args*, ***kwargs*)

instance_update_and_get_original(**args*, ***kwargs*)
Set the given properties on an instance and update it. Return a shallow copy of the original instance reference, as well as the updated one.

Parameters

- **context** – = request context object
- **instance_uuid** – = instance uuid
- **values** – = dict containing column values

If "expected_task_state" exists in values, the update can only happen when the task state before update matches expected_task_state. Otherwise a UnexpectedTaskStateError is thrown.

Returns a tuple of the form (old_instance_ref, new_instance_ref)

Raises NotFound if instance does not exist.

key_pair_count_by_user(*context*, *user_id*)

key_pair_create(**args*, ***kwargs*)

key_pair_destroy(**args*, ***kwargs*)

key_pair_get(**args*, ***kwargs*)

key_pair_get_all_by_user(**args*, ***kwargs*)

migration_create(*context*, *values*)

migration_get(*context*, *id*)

migration_get_all_by_filters(*context*, *filters*)

migration_get_by_instance_and_status(*context*, *instance_uuid*, *status*)

migration_get_in_progress_by_host_and_node(*context*, *host*, *node*)

migration_get_unconfirmed_by_dest_compute(*context*, *confirm_window*, *dest_compute*, *use_slave=False*)

migration_update(*context*, *id*, *values*)

model_query(*context*, *model*, *args=None*, *session=None*, *use_slave=False*, *read_deleted=None*, *project_only=False*)

Query helper that accounts for context's *read_deleted* field.

Parameters

- **context** – NovaContext of the query.
- **model** – Model to query. Must be a subclass of ModelBase.
- **args** – Arguments to query. If None - model is used.
- **session** – If present, the session to use.
- **use_slave** – If true, use a slave connection to the DB if creating a session.
- **read_deleted** – If not None, overrides context's read_deleted field. Permitted values are 'no', which does not return deleted values; 'only', which only returns deleted values; and 'yes', which does not filter deleted values.
- **project_only** – If set and context is user-type, then restrict query to match the context's project_id. If set to 'allow_none', restriction includes project_id = None.

network_associate(*context*, *project_id*, *network_id=None*, *force=False*)

Associate a project with a network.

called by project_get_networks under certain conditions and network manager add_network_to_project()

only associate if the project doesn't already have a network or if force is True

force solves race condition where a fresh project has multiple instance builds simultaneously picked up by multiple network hosts which attempt to associate the project with multiple networks force should only be used as a direct consequence of user request all automated requests should not use force

network_count_reserved_ips(*context*, *network_id*)

network_create_safe(*context*, *values*)

network_delete_safe(*context*, *network_id*)

network_disassociate(*context*, *network_id*, *disassociate_host*, *disassociate_project*)

network_get(**args*, ***kwargs*)

network_get_all(**args*, ***kwargs*)

network_get_all_by_host(*context*, *host*)

network_get_all_by_uuids(**args*, ***kwargs*)

network_get_associated_fixed_ips(*context*, *network_id*, *host=None*)

network_get_by_cidr(*context*, *cidr*)

network_get_by_uuid(*context*, *uuid*)

network_in_use_on_host(*context*, *network_id*, *host*)

network_set_host(**args*, ***kwargs*)

network_update(**args*, ***kwargs*)

not_equal(**values*)

pci_device_destroy(*context*, *node_id*, *address*)

pci_device_get_all_by_instance_uuid(**args*, ***kwargs*)

pci_device_get_all_by_node(*context*, *node_id*)

pci_device_get_by_addr(*context*, *node_id*, *dev_addr*)

pci_device_get_by_id(*context*, *id*)

pci_device_update(*context*, *node_id*, *address*, *values*)

process_sort_params(*sort_keys, sort_dirs, default_keys=['created_at', 'id'], default_dir='asc'*)

Process the sort parameters to include default keys.

Creates a list of sort keys and a list of sort directions. Adds the default keys to the end of the list if they are not already included.

When adding the default keys to the sort keys list, the associated direction is: 1) The first element in the 'sort_dirs' list (if specified), else 2) 'default_dir' value (Note that 'asc' is the default value since this is the default in sqlalchemy.utils.paginate_query)

Parameters

- **sort_keys** – List of sort keys to include in the processed list
- **sort_dirs** – List of sort directions to include in the processed list
- **default_keys** – List of sort keys that need to be included in the processed list, they are added at the end of the list if not already specified.
- **default_dir** – Sort direction associated with each of the default keys that are not supplied, used when they are added to the processed list

Returns list of sort keys, list of sort directions

Raises exception.InvalidInput If more sort directions than sort keys are specified or if an invalid sort direction is specified

project_get_networks(**args*, ***kwargs*)

provider_fw_rule_create(**args*, ***kwargs*)

provider_fw_rule_destroy(**args*, ***kwargs*)

provider_fw_rule_get_all(**args*, ***kwargs*)

quota_class_create(**args*, ***kwargs*)

quota_class_get(**args*, ***kwargs*)

quota_class_get_all_by_name(**args*, ***kwargs*)

quota_class_get_default(*context*)

quota_class_update(**args*, ***kwargs*)

quota_create(*context*, *project_id*, *resource*, *limit*, *user_id=None*)

quota_destroy_all_by_project(*context*, *project_id*)

quota_destroy_all_by_project_and_user(*context*, *project_id*, *user_id*)

quota_get(**args*, ***kwargs*)

quota_get_all(**args*, ***kwargs*)

quota_get_all_by_project(**args*, ***kwargs*)

quota_get_all_by_project_and_user(**args*, ***kwargs*)

quota_reserve(**args*, ***kwargs*)

quota_update(*context*, *project_id*, *resource*, *limit*, *user_id=None*)

quota_usage_get(**args*, ***kwargs*)

quota_usage_get_all_by_project(**args*, ***kwargs*)

quota_usage_get_all_by_project_and_user(**args*, ***kwargs*)

quota_usage_update(*context*, *project_id*, *user_id*, *resource*, ***kwargs*)

require_admin_context(*f*)
Decorator to require admin request context.

The first argument to the wrapped function must be the context.

require_aggregate_exists(*f*)
Decorator to require the specified aggregate to exist.

Requires the wrapped function to use context and aggregate_id as their first two arguments.

require_context(*f*)
Decorator to require *any* user or admin context.

This does no authorization for user or project access matching, see `nova.context.authorize_project_context()` and `nova.context.authorize_user_context()`.

The first argument to the wrapped function must be the context.

require_instance_exists_using_uuid(*f*)
Decorator to require the specified instance to exist.

Requires the wrapped function to use context and instance_uuid as their first two arguments.

reservation_commit(**args*, ***kwargs*)

reservation_expire(**args*, ***kwargs*)

reservation_rollback(**args*, ***kwargs*)

s3_image_create(*context*, *image_uuid*)
Create local s3 image represented by provided uuid.

s3_image_get(*context*, *image_id*)
Find local s3 image represented by the provided id.

s3_image_get_by_uuid(*context*, *image_uuid*)
Find local s3 image represented by the provided uuid.

security_group_create(**args*, ***kwargs*)

security_group_default_rule_create(*context*, *values*)

security_group_default_rule_destroy(*context*, *security_group_rule_default_id*)

security_group_default_rule_get(**args*, ***kwargs*)

security_group_default_rule_list(**args*, ***kwargs*)

security_group_destroy(**args*, ***kwargs*)

security_group_ensure_default(*context*)
Ensure default security group exists for a project_id.

security_group_get(**args*, ***kwargs*)

security_group_get_all(**args*, ***kwargs*)

security_group_get_by_instance(**args*, ***kwargs*)

security_group_get_by_name(**args*, ***kwargs*)

security_group_get_by_project(**args*, ***kwargs*)

security_group_in_use(**args*, ***kwargs*)

security_group_rule_count_by_group(**args*, ***kwargs*)

security_group_rule_create(**args*, ***kwargs*)

security_group_rule_destroy(**args*, ***kwargs*)

security_group_rule_get(**args*, ***kwargs*)

security_group_rule_get_by_security_group(**args*, ***kwargs*)

security_group_rule_get_by_security_group_grantee(**args*, ***kwargs*)

security_group_update(**args*, ***kwargs*)

service_create(*context*, *values*)

service_destroy(*context*, *service_id*)

service_get(*context*, *service_id*, *use_slave=False*)

service_get_all(*context*, *disabled=None*)

service_get_all_by_binary(*context*, *binary*)

service_get_all_by_host(*context*, *host*)

service_get_all_by_topic(*context*, *topic*)

service_get_by_compute_host(*context*, *host*, *use_slave=False*)

service_get_by_host_and_binary(*context*, *host*, *binary*)

service_get_by_host_and_topic(*context*, *host*, *topic*)

service_update(**args*, ***kwargs*)

task_log_begin_task(*context*, *task_name*, *period_beginning*, *period_ending*, *host*, *task_items=None*, *message=None*)

task_log_end_task(*context*, *task_name*, *period_beginning*, *period_ending*, *host*, *errors*, *message=None*)

task_log_get(*context*, *task_name*, *period_beginning*, *period_ending*, *host*, *state=None*)

task_log_get_all(*context*, *task_name*, *period_beginning*, *period_ending*, *host=None*, *state=None*)

virtual_interface_create(**args*, ***kwargs*)
Create a new virtual interface record in the database.

Parameters **values** – = dict containing column values

virtual_interface_delete_by_instance(**args*, ***kwargs*)
Delete virtual interface records that are associated with the instance given by instance_id.

Parameters **instance_uuid** – = uuid of instance

virtual_interface_get(**args*, ***kwargs*)
Gets a virtual interface from the table.

Parameters **vif_id** – = id of the virtual interface

virtual_interface_get_all(**args*, ***kwargs*)
Get all vifs.

virtual_interface_get_by_address(**args*, ***kwargs*)
Gets a virtual interface from the table.

Parameters **address** – = the address of the interface you're looking to get

virtual_interface_get_by_instance(**args*, ***kwargs*)
Gets all virtual interfaces for instance.

Parameters **instance_uuid** – = uuid of the instance to retrieve vifs for

virtual_interface_get_by_instance_and_network(**args*, ***kwargs*)
Gets virtual interface for instance that's associated with network.

virtual_interface_get_by_uuid(**args*, ***kwargs*)
Gets a virtual interface from the table.

Parameters **vif_uuid** – the uuid of the interface you're looking to get

vol_get_usage_by_time(**args*, ***kwargs*)
Return volumes usage that have been updated after a specified time.

vol_usage_update(**args*, ***kwargs*)

3.7.352 The nova.db.sqlalchemy.api_migrations.migrate_repo.versions.001_cell_m Module

upgrade(*migrate_engine*)

3.7.353 The nova.db.sqlalchemy.api_migrations.migrate_repo.versions.002_instar Module

upgrade(*migrate_engine*)

3.7.354 The nova.db.sqlalchemy.api_migrations.migrate_repo.versions.003_host_m Module

upgrade(*migrate_engine*)

3.7.355 The nova.db.sqlalchemy.api_models Module

class CellMapping(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Contains information on communicating with a cell

created_at

database_connection

id

name

transport_url

updated_at

uuid

class HostMapping(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

Contains mapping of a compute host to which cell it is in

cell_id

created_at

host

id

updated_at

class InstanceMapping(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

Contains the mapping of an instance to which cell it is in

cell_id

created_at

id

instance_uuid

project_id

updated_at

3.7.356 The nova.db.sqlalchemy.migrate_repo.manage Module

3.7.357 The nova.db.sqlalchemy.migrate_repo.versions.216_havana Module

Inet()

InetSmall()

MediumText()

upgrade(*migrate_engine*)

3.7.358 The nova.db.sqlalchemy.migrate_repo.versions.217_placeholder Module

upgrade(*migrate_engine*)

3.7.359 The nova.db.sqlalchemy.migrate_repo.versions.218_placeholder Module

upgrade(*migrate_engine*)

3.7.360 The nova.db.sqlalchemy.migrate_repo.versions.219_placeholder Module

upgrade(*migrate_engine*)

3.7.361 The nova.db.sqlalchemy.migrate_repo.versions.220_placeholder Module

upgrade(*migrate_engine*)

3.7.362 The nova.db.sqlalchemy.migrate_repo.versions.221_placeholder Module

upgrade(*migrate_engine*)

3.7.363 The nova.db.sqlalchemy.migrate_repo.versions.222_placeholder Module

upgrade(*migrate_engine*)

3.7.364 The nova.db.sqlalchemy.migrate_repo.versions.223_placeholder Module

upgrade(*migrate_engine*)

3.7.365 The nova.db.sqlalchemy.migrate_repo.versions.224_placeholder Module

upgrade(*migrate_engine*)

3.7.366 The nova.db.sqlalchemy.migrate_repo.versions.225_placeholder Module

upgrade(*migrate_engine*)

3.7.367 The nova.db.sqlalchemy.migrate_repo.versions.226_placeholder Module

upgrade(*migrate_engine*)

3.7.368 The nova.db.sqlalchemy.migrate_repo.versions.227_fix_project_user_quot Module

upgrade(*migrate_engine*)

3.7.369 The nova.db.sqlalchemy.migrate_repo.versions.228_add_metrics_in_comput Module

upgrade(*migrate_engine*)

3.7.370 The nova.db.sqlalchemy.migrate_repo.versions.229_add_extra_resources_i Module

upgrade(*migrate_engine*)

3.7.371 The nova.db.sqlalchemy.migrate_repo.versions.230_add_details_column_to Module

upgrade(*migrate_engine*)

3.7.372 The nova.db.sqlalchemy.migrate_repo.versions.231_add_ephemeral_key_uui Module

upgrade(*migrate_engine*)
Function adds ephemeral storage encryption key uuid field.

3.7.373 The nova.db.sqlalchemy.migrate_repo.versions.232_drop_dump_tables Module

upgrade(*migrate_engine*)

3.7.374 The nova.db.sqlalchemy.migrate_repo.versions.233_add_stats_in_compute_ Module

upgrade(*engine*)

3.7.375 The nova.db.sqlalchemy.migrate_repo.versions.234_add_expire_reservatio Module

upgrade(*migrate_engine*)

3.7.376 The nova.db.sqlalchemy.migrate_repo.versions.235_placeholder Module

upgrade(*migrate_engine*)

3.7.377 The nova.db.sqlalchemy.migrate_repo.versions.236_placeholder Module

upgrade(*migrate_engine*)

3.7.378 The nova.db.sqlalchemy.migrate_repo.versions.237_placeholder Module

upgrade(*migrate_engine*)

3.7.379 The nova.db.sqlalchemy.migrate_repo.versions.238_placeholder Module

upgrade(*migrate_engine*)

3.7.380 The nova.db.sqlalchemy.migrate_repo.versions.239_placeholder Module

upgrade(*migrate_engine*)

3.7.381 The nova.db.sqlalchemy.migrate_repo.versions.240_placeholder Module

upgrade(*migrate_engine*)

3.7.382 The nova.db.sqlalchemy.migrate_repo.versions.241_placeholder Module

upgrade(*migrate_engine*)

3.7.383 The nova.db.sqlalchemy.migrate_repo.versions.242_placeholder Module

upgrade(*migrate_engine*)

3.7.384 The nova.db.sqlalchemy.migrate_repo.versions.243_placeholder Module

upgrade(*migrate_engine*)

3.7.385 The nova.db.sqlalchemy.migrate_repo.versions.244_increase_user_id_leng Module

upgrade(*migrate_engine*)

3.7.386 The nova.db.sqlalchemy.migrate_repo.versions.245_add_mtu_and_dhcp_serv Module

upgrade (*migrate_engine*)
Function adds network mtu, dhcp_server, and share_dhcp fields.

3.7.387 The nova.db.sqlalchemy.migrate_repo.versions.246_add_compute_node_id_f Module

upgrade (*migrate_engine*)
Add missing foreign key constraint on pci_devices.compute_node_id.

3.7.388 The nova.db.sqlalchemy.migrate_repo.versions.247_nullable_mismatch Module

upgrade (*migrate_engine*)

3.7.389 The nova.db.sqlalchemy.migrate_repo.versions.248_add_expire_reservatio Module

upgrade (*migrate_engine*)

3.7.390 The nova.db.sqlalchemy.migrate_repo.versions.249_remove_duplicate_inde Module

upgrade (*migrate_engine*)
Remove duplicate index from block_device_mapping table.

3.7.391 The nova.db.sqlalchemy.migrate_repo.versions.250_remove_instance_group Module

upgrade (*migrate_engine*)
Remove the instance_group_metadata table.

3.7.392 The nova.db.sqlalchemy.migrate_repo.versions.251_add_numa_topology_to_ Module

upgrade (*migrate_engine*)

3.7.393 The nova.db.sqlalchemy.migrate_repo.versions.252_add_instance_extra_ta Module

upgrade (*migrate_engine*)

3.7.394 The nova.db.sqlalchemy.migrate_repo.versions.253_add_pci_requests_to_i Module

upgrade(*migrate_engine*)

3.7.395 The nova.db.sqlalchemy.migrate_repo.versions.254_add_request_id_in_pci Module

upgrade(*engine*)
Function adds request_id field.

3.7.396 The nova.db.sqlalchemy.migrate_repo.versions.255_placeholder Module

upgrade(*migrate_engine*)

3.7.397 The nova.db.sqlalchemy.migrate_repo.versions.256_placeholder Module

upgrade(*migrate_engine*)

3.7.398 The nova.db.sqlalchemy.migrate_repo.versions.257_placeholder Module

upgrade(*migrate_engine*)

3.7.399 The nova.db.sqlalchemy.migrate_repo.versions.258_placeholder Module

upgrade(*migrate_engine*)

3.7.400 The nova.db.sqlalchemy.migrate_repo.versions.259_placeholder Module

upgrade(*migrate_engine*)

3.7.401 The nova.db.sqlalchemy.migrate_repo.versions.260_placeholder Module

upgrade(*migrate_engine*)

3.7.402 The nova.db.sqlalchemy.migrate_repo.versions.261_placeholder Module

upgrade(*migrate_engine*)

3.7.403 The nova.db.sqlalchemy.migrate_repo.versions.262_placeholder Module

upgrade(*migrate_engine*)

3.7.404 The nova.db.sqlalchemy.migrate_repo.versions.263_placeholder Module

upgrade(*migrate_engine*)

3.7.405 The nova.db.sqlalchemy.migrate_repo.versions.264_placeholder Module

upgrade(*migrate_engine*)

3.7.406 The nova.db.sqlalchemy.migrate_repo.versions.265_remove_duplicated_inc Module

upgrade(*migrate_engine*)
Remove index that are subsets of other indexes.

3.7.407 The nova.db.sqlalchemy.migrate_repo.versions.266_add_instance_tags Module

upgrade(*migrate_engine*)

3.7.408 The nova.db.sqlalchemy.migrate_repo.versions.267_instance_uuid_non_nul Module

process_null_records(*meta*, *scan=True*)
Scans the database for null instance_uuid records for processing.

Parameters

- **meta** – sqlalchemy.MetaData object, assumes tables are reflected.
- **scan** – If True, does a query and fails the migration if NULL instance uuid entries found. If False, makes instances.uuid non-nullable.

scan_for_null_records(*table*, *col_name*, *check_fkeys*)
Queries the table looking for NULL instances of the given column.

Parameters

- **col_name** – The name of the column to look for in the table.
- **check_fkeys** – If True, check the table for foreign keys back to the instances table and if not found, return.

Raises exception.ValidationError: If any records are found.

upgrade(*migrate_engine*)

3.7.409 The nova.db.sqlalchemy.migrate_repo.versions.268_add_host_in_compute_r Module

upgrade(*migrate_engine*)

3.7.410 The nova.db.sqlalchemy.migrate_repo.versions.269_add_numa_node_column Module

upgrade(*migrate_engine*)

3.7.411 The nova.db.sqlalchemy.migrate_repo.versions.270_flavor_data_in_extra Module

upgrade(*migrate_engine*)

3.7.412 The nova.db.sqlalchemy.migrate_repo.versions.271_sqlite_postgresql_ind Module

ensure_index_exists(*migrate_engine*, *table_name*, *index_name*, *column_names*)

ensure_index_removed(*migrate_engine*, *table_name*, *index_name*)

upgrade(*migrate_engine*)
Add indexes missing on SQLite and PostgreSQL.

3.7.413 The nova.db.sqlalchemy.migrate_repo.versions.272_add_keypair_type Module

upgrade(*migrate_engine*)

3.7.414 The nova.db.sqlalchemy.migrate_repo.versions.273_sqlite_foreign_keys Module

upgrade(*migrate_engine*)

3.7.415 The nova.db.sqlalchemy.migrate_repo.versions.274_update_instances_proj Module

upgrade(*migrate_engine*)
Change instances (project_id) index to cover (project_id, deleted).

3.7.416 The nova.db.sqlalchemy.migrate_repo.versions.275_add_keypair_type Module

upgrade(*migrate_engine*)
Function adds key_pairs type field.

3.7.417 The nova.db.sqlalchemy.migrate_repo.versions.276_vcpu_model Module

upgrade(*migrate_engine*)

3.7.418 The nova.db.sqlalchemy.migrate_repo.versions.277_add_fixed_ip_updated_ Module

upgrade(*migrate_engine*)

3.7.419 The nova.db.sqlalchemy.migrate_repo.versions.278_remove_service_fk_in_ Module

upgrade(*migrate_engine*)

3.7.420 The nova.db.sqlalchemy.migrate_repo.versions.279_fix_unique_constraint Module

upgrade(*migrate_engine*)

3.7.421 The nova.db.sqlalchemy.migrate_repo.versions.280_add_nullable_false_to Module

upgrade(*migrate_engine*)
Function enforces non-null value for keypairs name field.

3.7.422 The nova.db.sqlalchemy.migrate_repo.versions.281_placeholder Module

upgrade(*migrate_engine*)

3.7.423 The nova.db.sqlalchemy.migrate_repo.versions.282_placeholder Module

upgrade(*migrate_engine*)

3.7.424 The nova.db.sqlalchemy.migrate_repo.versions.283_placeholder Module

upgrade(*migrate_engine*)

3.7.425 The nova.db.sqlalchemy.migrate_repo.versions.284_placeholder Module

upgrade(*migrate_engine*)

3.7.426 The nova.db.sqlalchemy.migrate_repo.versions.285_placeholder Module

upgrade(*migrate_engine*)

3.7.427 The nova.db.sqlalchemy.migrate_repo.versions.286_placeholder Module

upgrade(*migrate_engine*)

3.7.428 The nova.db.sqlalchemy.migrate_repo.versions.287_placeholder Module

upgrade(*migrate_engine*)

3.7.429 The nova.db.sqlalchemy.migrate_repo.versions.288_placeholder Module

upgrade(*migrate_engine*)

3.7.430 The nova.db.sqlalchemy.migrate_repo.versions.289_placeholder Module

upgrade(*migrate_engine*)

3.7.431 The nova.db.sqlalchemy.migrate_repo.versions.290_placeholder Module

upgrade(*migrate_engine*)

3.7.432 The nova.db.sqlalchemy.migrate_repo.versions.291_enforce_flavors_migra Module

upgrade(*migrate_engine*)

3.7.433 The nova.db.sqlalchemy.migrate_repo.versions.292_drop_nova_volumes_tab Module

upgrade(*engine*)

3.7.434 The nova.db.sqlalchemy.migrate_repo.versions.293_add_migration_type Module

upgrade(*migrate_engine*)

3.7.435 The nova.db.sqlalchemy.migrate_repo.versions.294_add_service_heartbeat Module

upgrade(*migrate_engine*)

3.7.436 The nova.db.sqlalchemy.migrate_repo.versions.295_add_virtual_interface Module

upgrade(*migrate_engine*)

3.7.437 The nova.db.sqlalchemy.migrate_repo.versions.296_add_missing_db2_fkeys Module

upgrade(*migrate_engine*)

3.7.438 The nova.db.sqlalchemy.migration Module

class AddColumn(*table_name*, *column*, *desired_phase=None*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

class AddForeignKey(*fkc*, *desired_phase=None*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

class AddIndex(*index*, *args*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

class AddTable(*table*)
Bases: nova.db.sqlalchemy.migration.OperationBase

desired_phase = 'expand'

execute(*ddlop*)

class AddUniqueConstraint(*uc*, *desired_phase=None*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

class AlterColumn(*table_name*, *column_name*, *args*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

class Converter
Bases: object

convert_alembic(*diffs*)

class DropColumn(*table_name*, *column*)
Bases: nova.db.sqlalchemy.migration.OperationBase

desired_phase = 'contract'

execute(*ddlop*)

removes = True

class **DropForeignKey**(*fkc*, *desired_phase=None*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

removes = True

class **DropIndex**(*index*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

removes = True

class **DropTable**(*table*)
Bases: nova.db.sqlalchemy.migration.OperationBase

desired_phase = 'contract'

execute(*ddlop*)

removes = True

class **DropUniqueConstraint**(*uc*)
Bases: nova.db.sqlalchemy.migration.OperationBase

execute(*ddlop*)

removes = True

class **OperationBase**
Bases: object

desired_phase = 'migrate'

removes = False

class **Scheduler**(*ops=None*)
Bases: object

add(*op*)

add_edge(*f*, *t*)

handle_conflicts(*metadata*)

order_drop_add()

schedule()

sort()

db_contract(*dryrun=False*, *database='main'*)

db_expand(*dryrun=False*, *database='main'*)

db_initial_version(*database='main'*)

db_migrate(*dryrun=False*, *database='main'*)

db_null_instance_uuid_scan(*delete=False*)
Scans the database for NULL instance_uuid records.

Parameters **delete** – If true, delete NULL instance_uuid records found, else just query to see if they exist for reporting.

Returns dict of table name to number of hits for NULL instance_uuid rows.

db_sync(*version=None*, *database='main'*)

db_version(*database='main'*)

db_version_control(*version=None*, *database='main'*)

get_engine(*database='main'*)

3.7.439 The nova.db.sqlalchemy.models Module

SQLAlchemy models for nova data.

class AgentBuild(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents an agent build.

architecture

created_at

deleted

deleted_at

hypervisor

id

md5hash

os

updated_at

url

version

class Aggregate(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a cluster of hosts that exists in this zone.

availability_zone

created_at

deleted

deleted_at

hosts

id

metadetails

name

updated_at

class AggregateHost(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a host that is member of an aggregate.

aggregate_id

created_at

deleted

deleted_at

host

id

updated_at

class AggregateMetadata(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a metadata key/value pair for an aggregate.

aggregate_id

created_at

deleted

deleted_at

id

key

updated_at

value

class BandwidthUsage(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Cache for instance bandwidth usage data pulled from the hypervisor.

bw_in

bw_out

created_at

deleted

deleted_at

id

last_ctr_in

last_ctr_out

last_refreshed

mac

start_period

updated_at

uuid

class BlockDeviceMapping(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents block device mapping that is defined by EC2.

boot_index

connection_info

created_at

delete_on_termination

deleted

deleted_at

destination_type

device_name

device_type

disk_bus

guest_format

id

image_id

instance

instance_uuid

no_device

snapshot_id

source_type

updated_at

volume_id

volume_size

class Cell(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents parent and child cells of this cell. Cells can have multiple parents and children, so there could be any number of entries with is_parent=True or False

api_url

created_at

deleted

deleted_at

id

is_parent

name

transport_url

updated_at

weight_offset

weight_scale

class Certificate(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a x509 certificate.

created_at

deleted

deleted_at

file_name

id

project_id

updated_at

user_id

class ComputeNode(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a running compute service on a host.

cpu_info

created_at

current_workload

deleted

deleted_at

disk_available_least

extra_resources

free_disk_gb

free_ram_mb

host

host_ip

hypervisor_hostname

hypervisor_type

hypervisor_version

id

local_gb

local_gb_used

memory_mb

memory_mb_used

metrics

numa_topology

pci_stats

running_vms

service_id

stats

supported_instances

updated_at

vcpus

vcpus_used

class Console(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a console session for an instance.

created_at

deleted

deleted_at

id

instance_name

instance_uuid

password

pool

pool_id

port

updated_at

class ConsolePool(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents pool of consoles on the same physical node.

address

compute_host

console_type

created_at

deleted

deleted_at

host

id

password

public_hostname

updated_at

username

class DNSDomain (***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a DNS domain with availability zone or project info.

availability_zone

created_at

deleted

deleted_at

domain

project_id

scope

updated_at

class FixedIp (***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a fixed ip for an instance.

address

allocated

created_at

deleted

deleted_at

host

id

instance

instance_uuid

leased

network

network_id

reserved

updated_at

virtual_interface

virtual_interface_id

class FloatingIp (***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a floating ip that dynamically forwards to a fixed ip.

address

auto_assigned

created_at

deleted

deleted_at

fixed_ip

fixed_ip_id

host

id

interface

pool

project_id

updated_at

class Instance (***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a guest VM.

access_ip_v4

access_ip_v6

architecture

auto_disk_config

availability_zone

cell_name

cleaned

config_drive

created_at

default_ephemeral_device

default_swap_device

deleted

deleted_at

disable_terminate

display_description

display_name

ephemeral_gb

ephemeral_key_uuid

host

hostname

id

image_ref

injected_files = []

instance_type_id

internal_id

kernel_id

key_data

key_name

launch_index

launched_at

launched_on

locked

locked_by

memory_mb

name

node

os_type

power_state

progress

project_id

ramdisk_id

reservation_id

root_device_name

root_gb

scheduled_at

shutdown_terminate

task_state

terminated_at

updated_at

user_data

user_id

uuid

vcpus

vm_mode

vm_state

class InstanceAction(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Track client actions on an instance.

The intention is that there will only be one of these per user request. A lookup by (instance_uuid, request_id) should always return a single result.

action

created_at

deleted

deleted_at

finish_time

id

instance_uuid

message

project_id

request_id

start_time

updated_at

user_id

class InstanceActionEvent(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Track events that occur during an InstanceAction.

action_id

created_at

deleted

deleted_at

details

event

finish_time

host

id

result

start_time

traceback

updated_at

class InstanceExtra(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

created_at

deleted

deleted_at

flavor

id

instance

instance_uuid

numa_topology

pci_requests

updated_at

vcpu_model

class InstanceFault(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

code

created_at

deleted

deleted_at

details

host

id

instance_uuid

message

updated_at

class InstanceGroup(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents an instance group.

A group will maintain a collection of instances and the relationship between them.

created_at

deleted

deleted_at

id

members

name

policies

project_id

updated_at

user_id

uuid

class InstanceGroupMember(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents the members for an instance group.

created_at

deleted

deleted_at

group_id

id

instance_id

updated_at

class InstanceGroupPolicy(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents the policy type for an instance group.

created_at

deleted

deleted_at

group_id

id

policy

updated_at

class InstanceIdMapping(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Compatibility layer for the EC2 instance service.

created_at

deleted

deleted_at

id

updated_at

uuid

class InstanceInfoCache(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a cache of information about an instance

created_at

deleted

deleted_at

id

instance

instance_uuid

network_info

updated_at

class InstanceMetadata(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a user-provided metadata key/value pair for an instance.

created_at

deleted

deleted_at

id

instance

instance_uuid

key

updated_at

value

class InstanceSystemMetadata(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a system-owned metadata key/value pair for an instance.

created_at

deleted

deleted_at

id

instance

instance_uuid

key

updated_at

value

class InstanceTypeExtraSpecs(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents additional specs as key/value pairs for an instance_type.

created_at

deleted

deleted_at

id

instance_type

instance_type_id

key

updated_at

value

class **InstanceTypeProjects**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represent projects associated instance_types.

created_at

deleted

deleted_at

id

instance_type

instance_type_id

project_id

updated_at

class **InstanceTypes**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents possible flavors for instances.

Note: instance_type and flavor are synonyms and the term instance_type is deprecated and in the process of being removed.

created_at

deleted

deleted_at

disabled

ephemeral_gb

flavorid

id

is_public

memory_mb

name

root_gb

rxtx_factor

swap

updated_at

vcpu_weight

vcpus

class **KeyPair**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a public key pair for ssh / WinRM.

created_at

deleted

deleted_at

fingerprint

id

name

public_key

type

updated_at

user_id

MediumText()

class **Migration**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a running host-to-host migration.

created_at

deleted

deleted_at

dest_compute

dest_host

dest_node

hidden

id

instance

instance_uuid

migration_type

new_instance_type_id

old_instance_type_id

source_compute

source_node

status

updated_at

class **Network**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a network.

bridge

bridge_interface

broadcast

cidr

cidr_v6

created_at

deleted

deleted_at

dhcp_server

dhcp_start

dns1

dns2

enable_dhcp

gateway

gateway_v6

host

id

injected

label

mtu

multi_host

netmask

netmask_v6

priority

project_id

rxtx_base

share_address

updated_at

uuid

vlan

vpn_private_address

vpn_public_address

vpn_public_port

class NovaBase

Bases: `oslo_db.sqlalchemy.models.SoftDeleteMixin`, `oslo_db.sqlalchemy.models.TimestampMixin`, `oslo_db.sqlalchemy.models.ModelBase`

metadata = None

save(*session=None*)

class PciDevice(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `nova.db.sqlalchemy.models.NovaBase`

Represents a PCI host device that can be passed through to instances.

address

compute_node_id

created_at

deleted

deleted_at

dev_id

dev_type

extra_info

id

instance

instance_uuid

label

numa_node

product_id

request_id

status

updated_at

vendor_id

class ProjectUserQuota(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a single quota override for a user with in a project.

created_at

deleted

deleted_at

hard_limit

id

project_id

resource

uniq_name = 'uniq_project_user_quotas0user_id0project_id0resource0deleted'

updated_at

user_id

class ProviderFirewallRule(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a rule in a security group.

cidr

created_at

deleted

deleted_at

from_port

id

protocol

to_port

updated_at

class Quota(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a single quota override for a project.

If there is no row for a given project id and resource, then the default for the quota class is used. If there is no row for a given quota class and resource, then the default for the deployment is used. If the row is present but the hard limit is Null, then the resource is unlimited.

created_at

deleted

deleted_at

hard_limit

id

project_id

resource

updated_at

class QuotaClass(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a single quota override for a quota class.

If there is no row for a given quota class and resource, then the default for the deployment is used. If the row is present but the hard limit is Null, then the resource is unlimited.

class_name

created_at

deleted

deleted_at

hard_limit

id

resource

updated_at

class QuotaUsage(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents the current usage for a given resource.

created_at

deleted

deleted_at

id

in_use

project_id

reserved

resource

total

until_refresh

updated_at

user_id

class Reservation(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a resource reservation for quotas.

created_at

deleted

deleted_at

delta

expire

id

project_id

resource

updated_at

usage

usage_id

user_id

uuid

class S3Image(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Compatibility layer for the S3 image service talking to Glance.

created_at

deleted

deleted_at

id

updated_at

uuid

class SecurityGroup(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a security group.

created_at

deleted

deleted_at

description

id

instances

name

project_id

updated_at

user_id

class SecurityGroupIngressDefaultRule(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

cidr

created_at

deleted

deleted_at

from_port

id

protocol

to_port

updated_at

class SecurityGroupIngressRule(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a rule in a security group.

cidr

created_at

deleted

deleted_at

from_port

grantee_group

group_id

id

parent_group

parent_group_id

protocol

to_port

updated_at

class SecurityGroupInstanceAssociation(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

created_at

deleted

deleted_at

id

instance_uuid

security_group_id

updated_at

class Service(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a running service on a host.

binary

created_at

deleted

deleted_at

disabled

disabled_reason

host

id

last_seen_up

report_count

topic

updated_at

class Snapshot(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a block storage device that can be attached to a VM.

created_at

deleted

deleted_at

display_description

display_name

id

name

progress

project_id

scheduled_at

status

updated_at

user_id

volume_id

volume_name

volume_size

class SnapshotIdMapping(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Compatibility layer for the EC2 snapshot service.

created_at

deleted

deleted_at

id

updated_at

uuid

class Tag(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, oslo_db.sqlalchemy.models.ModelBase

Represents the tag for a resource.

instance

resource_id

tag

class TaskLog(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Audit log for background periodic tasks.

created_at

deleted

deleted_at

errors

host

id

message

period_beginning

period_ending

state

task_items

task_name

updated_at

class VirtualInterface(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Represents a virtual interface on an instance.

address

created_at

deleted

deleted_at

id

instance_uuid

network_id

updated_at

uuid

class VolumeIdMapping(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Compatibility layer for the EC2 volume service.

created_at

deleted

deleted_at

id

updated_at

uuid

class VolumeUsage(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, nova.db.sqlalchemy.models.NovaBase

Cache for volume usage data pulled from the hypervisor.

availability_zone

created_at

curr_last_refreshed

curr_read_bytes

curr_reads

curr_write_bytes

curr_writes

deleted

deleted_at

id

instance_uuid

project_id

tot_last_refreshed

tot_read_bytes

tot_reads

tot_write_bytes

tot_writes

updated_at

user_id

volume_id

3.7.440 The nova.db.sqlalchemy.types Module

Custom SQLAlchemy types.

class CIDR(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

An SQLAlchemy type representing a CIDR definition.

impl
alias of String

load_dialect_impl(*dialect*)

process_bind_param(*value*, *dialect*)
Process/Formats the value before insert it into the db.

process_result_value(*value*, *dialect*)

class IPAddress(**args*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

An SQLAlchemy type representing an IP-address.

impl
alias of String

load_dialect_impl(*dialect*)

process_bind_param(*value*, *dialect*)
Process/Formats the value before insert it into the db.

3.7.441 The nova.db.sqlalchemy.utils Module

class DeleteFromSelect(*table*, *select*, *column*)
Bases: sqlalchemy.sql.dml.UpdateBase

check_shadow_table(*migrate_engine*, *table_name*)
This method checks that table with table_name and corresponding shadow table have same columns.

create_shadow_table(*migrate_engine*, *table_name=None*, *table=None*, ***col_name_col_instance*)
This method create shadow table for table with name table_name or table instance table. :param table_name: Autoload table with this name and create shadow table :param table: Autoloaded table, so just create corresponding shadow table. :param col_name_col_instance: contains pair column_name=column_instance.

column_instance is instance of Column. These params are required only for columns that have unsupported types by sqlite. For example BigInteger. :returns: The created shadow_table object.

visit_delete_from_select(*element*, *compiler*, ***kw*)

3.7.442 The nova.debugger Module

enabled()

init()

register_cli_opts()

3.7.443 The nova.exception Module

Nova base exception handling.

Includes decorator for re-raising Nova-type exceptions.

SHOULD include dedicated exception logging.

exception AddressOutOfRange(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(address)s is not within %(cidr)s.'

exception AdminRequired(*message=None*, ***kwargs*)
Bases: nova.exception.Forbidden

msg_fmt = u'User does not have admin privileges'

exception AgentBuildExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Agent-build with hypervisor %(hypervisor)s os %(os)s architecture %(architecture)s exists.'

exception AgentBuildNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'No agent-build associated with id %(id)s.'

exception AgentError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Error during following call to agent: %(method)s'

exception AgentNotImplemented(*message=None*, ***kwargs*)
Bases: nova.exception.AgentError

msg_fmt = u'Agent does not support the call: %(method)s'

exception AgentTimeout(*message=None*, ***kwargs*)
Bases: nova.exception.AgentError

msg_fmt = u'Unable to contact guest agent. The following call timed out: %(method)s'

exception AggregateError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u"Aggregate %(aggregate_id)s: action '%(action)s' caused an error: %(reason)s."

exception AggregateHostExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Aggregate %(aggregate_id)s already has host %(host)s.'

exception AggregateHostNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Aggregate %(aggregate_id)s has no host %(host)s.'

exception AggregateMetadataNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Aggregate %(aggregate_id)s has no metadata with key %(metadata_key)s.'

exception AggregateNameExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Aggregate %(aggregate_name)s already exists.'

exception AggregateNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Aggregate %(aggregate_id)s could not be found.'

exception AutoDiskConfigDisabledByImage(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Requested image %(image)s has automatic disk resize disabled.'

exception BDMNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'No Block Device Mapping with id %(id)s.'

exception Base64Exception(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Invalid Base 64 data for file %(path)s'

exception BuildAbortException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Build of instance %(instance_uuid)s aborted: %(reason)s'

exception CPUPinningInvalid(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Cannot pin/unpin cpus %(requested)s from the following pinned set %(pinned)s'

exception CPUPinningNotSupported(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'CPU pinning is not supported by the host: %(reason)s'

exception CannotDisassociateAutoAssignedFloatingIP(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

ec2_code = 'UnsupportedOperation'

msg_fmt = u'Cannot disassociate auto assigned floating ip'

exception CannotResizeDisk(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Server disk was unable to be resized because: %(reason)s'

exception CannotResizeToSameFlavor(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'When resizing, instances must change flavor!'

exception CellExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Cell with name %(name)s already exists.'

exception CellMappingNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Cell %(uuid)s has no mapping.'

exception CellMaxHopCountReached(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Cell message has reached maximum hop count: %(hop_count)s'

exception CellNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u"Cell %(cell_name)s doesn't exist."

exception CellRoutingInconsistency(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Inconsistency in cell routing: %(reason)s'

exception CellServiceAPIMethodNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Service API method not found: %(detail)s'

exception CellTimeout(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Timeout waiting for response from cell'

exception CellsUpdateUnsupported(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Cannot update cells configuration file.'

exception CidrConflict(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 409

msg_fmt = u'Requested cidr (%(cidr)s) conflicts with existing cidr (%(other)s)'

exception CinderConnectionFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Connection to cinder host failed: %(reason)s'

exception ClassNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Class %(class_name)s could not be found: %(exception)s'

exception ComputeHostMetricNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Metric %(name)s could not be found on the compute host node %(host)s.%(node)s.'

exception ComputeHostNotCreated(*message=None*, ***kwargs*)
Bases: nova.exception.HostNotFound

msg_fmt = u'Compute host %(name)s needs to be created first before updating.'

exception ComputeHostNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.HostNotFound

msg_fmt = u'Compute host %(host)s could not be found.'

exception ComputeResourcesUnavailable(*message=None*, ***kwargs*)
Bases: nova.exception.ServiceUnavailable

msg_fmt = u'Insufficient compute resources: %(reason)s.'

exception ComputeServiceInUse(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Compute service of %(host)s is still in use.'

exception ComputeServiceUnavailable(*message=None*, ***kwargs*)
Bases: nova.exception.ServiceUnavailable

msg_fmt = u'Compute service of %(host)s is unavailable at this time.'

exception ConfigDriveInvalidValue(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid value for Config Drive option: %(option)s'

exception ConfigDriveMountFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Could not mount vfat config drive. %(operation)s failed. Error: %(error)s'

exception ConfigDriveUnknownFormat(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Unknown config drive format %(format)s for %(os_type)s os type. Valid options are: iso9660, vfat or None

exception ConfigDriveUnsupportedFormat(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Unsupported config drive format %(format)s for %(image_type)s image type. Image path: %(image_path)

exception ConfigNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Could not find config at %(path)s'

exception ConsoleNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Console %(console_id)s could not be found.'

exception ConsoleNotFoundForInstance(*message=None*, ***kwargs*)
Bases: nova.exception.ConsoleNotFound

msg_fmt = u'Console for instance %(instance_uuid)s could not be found.'

exception ConsoleNotFoundInPoolForInstance(*message=None*, ***kwargs*)
Bases: nova.exception.ConsoleNotFound

msg_fmt = u'Console for instance %(instance_uuid)s in pool %(pool_id)s could not be found.'

exception ConsolePoolExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Console pool with host %(host)s, console_type %(console_type)s and compute_host %(compute_host)s alre

exception ConsolePoolNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Console pool %(pool_id)s could not be found.'

exception ConsolePoolNotFoundForHostType(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Console pool of type %(console_type)s for compute host %(compute_host)s on proxy host %(host)s not fou

exception ConsolePortRangeExhausted(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'The console port range %(min_port)d-%(max_port)d is exhausted.'

exception ConsoleTypeInvalid(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid console type %(console_type)s'

exception ConsoleTypeUnavailable(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Unavailable console type %(console_type)s.'

exception ConstraintNotMet(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 412

msg_fmt = u'Constraint not met.'

exception ConvertedException(*code=0*, *title=''*, *explanation=''*)
Bases: webob.exc.WSGIHTTPException

exception CoreAPIMissing(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Core API extensions are missing: %(missing_apis)s'

exception CouldNotFetchImage(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Could not fetch image %(image_id)s'

exception CouldNotUploadImage(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Could not upload image %(image_id)s'

exception CryptoCAFileNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.FileNotFound

msg_fmt = u'The CA file for %(project)s could not be found'

exception CryptoCRLFileNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.FileNotFound

msg_fmt = u'The CRL file for %(project)s could not be found'

exception DBNotAllowed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'%(binary)s attempted direct database access which is not allowed by policy'

exception DatabaseMigrationError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Database migration failed: %(reason)s'

exception DatastoreNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Could not find the datastore reference(s) which the VM uses.'

exception DecryptionFailure(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to decrypt text: %(reason)s'

exception DestinationDiskExists(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The supplied disk path (%(path)s) already exists, it is expected not to exist.'

exception DestinationHypervisorTooOld(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The instance requires a newer hypervisor version than has been provided.'

exception DeviceIsBusy(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The supplied device (%(device)s) is busy.'

exception DevicePathInUse(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

code = 409

msg_fmt = u'The supplied device path (%(path)s) is in use.'

exception DiskInfoReadWriteFail(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to read or write disk info file: %(reason)s'

exception DiskNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'No disk at %(location)s'

exception DuplicateVlan(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 409

msg_fmt = u'Detected existing vlan with id %(vlan)d'

exception EncryptionFailure(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to encrypt text: %(reason)s'

exception EnumFieldInvalid(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(typename)s in %(fieldname)s is not an instance of Enum'

exception EnumFieldUnset(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(fieldname)s missing field type'

exception ExternalNetworkAttachForbidden (*message=None*, ***kwargs*)
Bases: nova.exception.Forbidden

msg_fmt = u'It is not allowed to create an interface on external network %(network_uuid)s'

exception FileNotFound (*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'File %(file_path)s could not be found.'

exception FixedIpAlreadyInUse (*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Fixed IP address %(address)s is already in use on instance %(instance_uuid)s.'

exception FixedIpAssociateFailed (*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Fixed IP associate failed for network: %(net)s.'

exception FixedIpAssociatedWithMultipleInstances (*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'"More than one instance is associated with fixed ip address '%(address)s'."

exception FixedIpExists (*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Fixed ip %(address)s already exists.'

exception FixedIpInvalid (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Fixed IP address %(address)s is invalid.'

exception FixedIpLimitExceeded (*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Maximum number of fixed ips exceeded'

exception FixedIpNotFound (*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'No fixed IP associated with id %(id)s.'

exception FixedIpNotFoundForAddress (*message=None*, ***kwargs*)
Bases: nova.exception.FixedIpNotFound

msg_fmt = u'Fixed ip not found for address %(address)s.'

exception FixedIpNotFoundForInstance (*message=None*, ***kwargs*)
Bases: nova.exception.FixedIpNotFound

msg_fmt = u'Instance %(instance_uuid)s has zero fixed ips.'

exception FixedIpNotFoundForNetwork (*message=None*, ***kwargs*)
Bases: nova.exception.FixedIpNotFound

msg_fmt = u'Fixed IP address (%(address)s) does not exist in network (%(network_uuid)s).'

exception FixedIpNotFoundForNetworkHost (*message=None*, ***kwargs*)
Bases: nova.exception.FixedIpNotFound

msg_fmt = u'Network host %(host)s has zero fixed ips in network %(network_id)s.'

exception FixedIpNotFoundForSpecificInstance (*message=None*, ***kwargs*)
Bases: nova.exception.FixedIpNotFound

msg_fmt = u"Instance %(instance_uuid)s doesn't have fixed ip '%(ip)s'."

exception FlavorAccessExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Flavor access already exists for flavor %(flavor_id)s and project %(project_id)s combination.'

exception FlavorAccessNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Flavor access not found for %(flavor_id)s / %(project_id)s combination.'

exception FlavorCreateFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Unable to create flavor'

exception FlavorDiskTooSmall(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u"Flavor's disk is too small for requested image."

exception FlavorExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Flavor with name %(name)s already exists.'

exception FlavorExtraSpecUpdateCreateFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Flavor %(id)d extra spec cannot be updated or created after %(retries)d retries.'

exception FlavorExtraSpecsNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Flavor %(flavor_id)s has no extra specs with key %(extra_specs_key)s.'

exception FlavorIdExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Flavor with ID %(flavor_id)s already exists.'

exception FlavorMemoryTooSmall(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u"Flavor's memory is too small for requested image."

exception FlavorNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Flavor %(flavor_id)s could not be found.'

exception FlavorNotFoundByName(*message=None*, ***kwargs*)
Bases: nova.exception.FlavorNotFound

msg_fmt = u'Flavor with name %(flavor_name)s could not be found.'

exception FloatingIpAllocateFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Floating IP allocate failed.'

exception FloatingIpAssociateFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Floating IP %(address)s association has failed.'

exception `FloatingIpAssociated`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

ec2_code = 'UnsupportedOperation'

msg_fmt = u'Floating ip %(address)s is associated.'

exception `FloatingIpDNSExists`(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The DNS entry %(name)s already exists in domain %(domain)s.'

exception `FloatingIpExists`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Floating ip %(address)s already exists.'

exception `FloatingIpLimitExceeded`(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Maximum number of floating ips exceeded'

exception `FloatingIpMultipleFoundForAddress`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Multiple floating ips are found for address %(address)s.'

exception `FloatingIpNotAssociated`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Floating ip %(address)s is not associated.'

exception `FloatingIpNotFound`(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'UnsupportedOperation'

msg_fmt = u'Floating ip not found for id %(id)s.'

exception `FloatingIpNotFoundForAddress`(*message=None*, ***kwargs*)
Bases: nova.exception.FloatingIpNotFound

msg_fmt = u'Floating ip not found for address %(address)s.'

exception `FloatingIpNotFoundForHost`(*message=None*, ***kwargs*)
Bases: nova.exception.FloatingIpNotFound

msg_fmt = u'Floating ip not found for host %(host)s.'

exception `FloatingIpPoolNotFound`(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Floating ip pool not found.'

safe = True

exception `Forbidden`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 403

ec2_code = 'AuthFailure'

msg_fmt = u'Not authorized.'

exception `GlanceConnectionFailed`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Connection to glance host %(host)s:%(port)s failed: %(reason)s'

exception HostBinaryNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Could not find binary %(binary)s on host %(host)s.'

exception HostMappingNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Host '%(name)s' is not mapped to any cell"

exception HostNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Host %(host)s could not be found.'

exception HypervisorUnavailable(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Connection to the hypervisor is broken on host: %(host)s'

exception ImageCPUPinningForbidden(*message=None*, ***kwargs*)
Bases: nova.exception.Forbidden

msg_fmt = u"Image property 'hw_cpu_policy' is not permitted to override CPU pinning policy set against the flavor"

exception ImageDownloadModuleConfigurationError(*message=None*, ***kwargs*)
Bases: nova.exception.ImageDownloadModuleError

msg_fmt = u'The module %(module)s is misconfigured: %(reason)s.'

exception ImageDownloadModuleError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'There was an error with the download module %(module)s. %(reason)s'

exception ImageDownloadModuleMetaDataError(*message=None*, ***kwargs*)
Bases: nova.exception.ImageDownloadModuleError

msg_fmt = u'The metadata for this location will not work with this module %(module)s. %(reason)s.'

exception ImageDownloadModuleNotImplementedError(*message=None*, ***kwargs*)
Bases: nova.exception.ImageDownloadModuleError

msg_fmt = u'The method %(method_name)s is not implemented.'

exception ImageNUMATopologyAsymmetric(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Asymmetric NUMA topologies require explicit assignment of CPUs and memory to nodes in image or flavo

exception ImageNUMATopologyCPUDuplicates(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'CPU number %(cpunum)d is assigned to two nodes'

exception ImageNUMATopologyCPUOutOfRange(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'CPU number %(cpunum)d is larger than max %(cpumax)d'

exception ImageNUMATopologyCPUsUnassigned(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'CPU number %(cpuset)s is not assigned to any node'

exception ImageNUMATopologyForbidden(*message=None*, ***kwargs*)
Bases: nova.exception.Forbidden

msg_fmt = u"Image property '%(name)s' is not permitted to override NUMA configuration set against the flavor"

exception ImageNUMATopologyIncomplete(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'CPU and memory allocation must be provided for all NUMA nodes'

exception ImageNUMATopologyMemoryOutOfRange(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(memsize)d MB of memory assigned, but expected %(memtotal)d MB'

exception ImageNotActive(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

ec2_code = 'IncorrectState'

msg_fmt = u'Image %(image_id)s is not active.'

exception ImageNotAuthorized(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Not authorized for image %(image_id)s.'

exception ImageNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Image %(image_id)s could not be found.'

exception ImageNotFoundEC2(*message=None*, ***kwargs*)
Bases: nova.exception.ImageNotFound

msg_fmt = u'Image %(image_id)s could not be found. The nova EC2 API assigns image ids dynamically when they are li

exception ImageSerialPortNumberExceedFlavorValue(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Forbidden to exceed flavor value of number of serial ports passed in image meta.'

exception ImageSerialPortNumberInvalid(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Number of serial ports '%(num_ports)s' specified in '%(property)s' isn't valid."

exception ImageUnacceptable(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Image %(image_id)s is unacceptable: %(reason)s'

exception ImageVCPULimitsRangeExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Image vCPU limits %(sockets)d:%(cores)d:%(threads)d exceeds permitted %(maxsockets)d:%(maxcores)

exception ImageVCPULimitsRangeImpossible(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Requested vCPU limits %(sockets)d:%(cores)d:%(threads)d are impossible to satisfy for vcpus count %(vc

exception ImageVCPUTopologyRangeExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Image vCPU topology %(sockets)d:%(cores)d:%(threads)d exceeds permitted %(maxsockets)d:%(maxcor

exception **IncompatibleObjectVersion**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Version %(objver)s of %(objname)s is not supported. The maximum supported version is: %(supported)s'

exception **InstanceActionEventNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Event %(event)s not found for action id %(action_id)s'

exception **InstanceActionNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Action for request_id %(request_id)s on instance %(instance_uuid)s not found'

exception **InstanceDeployFailure**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to deploy instance: %(reason)s'

exception **InstanceExists**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Instance %(name)s already exists.'

exception **InstanceFaultRollback**(*inner_exception=None*)
Bases: nova.exception.NovaException

exception **InstanceGroupIdExists**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Instance group %(group_uuid)s already exists.'

exception **InstanceGroupMemberNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Instance group %(group_uuid)s has no member with id %(instance_id)s.'

exception **InstanceGroupNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Instance group %(group_uuid)s could not be found.'

exception **InstanceGroupPolicyNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Instance group %(group_uuid)s has no policy %(policy)s.'

exception **InstanceInfoCacheNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Info cache for instance %(instance_uuid)s could not be found.'

exception **InstanceInvalidState**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance %(instance_uuid)s in %(attr)s %(state)s. Cannot %(method)s while the instance is in this state.'

exception **InstanceIsLocked**(*message=None*, ***kwargs*)
Bases: nova.exception.InstanceInvalidState

msg_fmt = u'Instance %(instance_uuid)s is locked'

exception **InstanceMappingNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Instance %(uuid)s has no mapping to a cell.'

exception InstanceNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'InvalidInstanceID.NotFound'

msg_fmt = u'Instance %(instance_id)s could not be found.'

exception InstanceNotInRescueMode(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance %(instance_id)s is not in rescue mode'

exception InstanceNotReady(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance %(instance_id)s is not ready'

exception InstanceNotRescuable(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance %(instance_id)s cannot be rescued: %(reason)s'

exception InstanceNotRunning(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance %(instance_id)s is not running.'

exception InstancePasswordSetFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to set admin password on %(instance)s because %(reason)s'

safe = True

exception InstancePowerOffFailure(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to power off instance: %(reason)s'

exception InstancePowerOnFailure(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to power on instance: %(reason)s'

exception InstanceQuiesceNotSupported(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Quiescing is not supported in instance %(instance_id)s: %(reason)s'

exception InstanceRebootFailure(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to reboot instance: %(reason)s'

exception InstanceRecreateNotSupported(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance recreate is not supported.'

exception InstanceResumeFailure(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to resume instance: %(reason)s'

exception InstanceSuspendFailure(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to suspend instance: %(reason)s'

exception InstanceTagNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u"Instance %(instance_id)s has no tag '%(tag)s'"

exception InstanceTerminationFailure(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to terminate instance: %(reason)s'

exception InstanceUnacceptable(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Instance %(instance_id)s is unacceptable: %(reason)s'

exception InstanceUnknownCell(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Cell is not known for instance %(instance_uuid)s'

exception InstanceUserDataMalformed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'User data needs to be valid base 64.'

exception InstanceUserDataTooLarge(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'User data too large. User data must be no larger than %(maxsize)s bytes once base64 encoded. Your data is

exception InsufficientFreeMemory(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Insufficient free memory on compute node to start %(uuid)s.'

exception InterfaceAttachFailed(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to attach network adapter device to %(instance_uuid)s'

exception InterfaceDetachFailed(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to detach network adapter device from %(instance_uuid)s'

exception InternalError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

ec2_code = 'InternalError'

msg_fmt = '%(err)s'

exception Invalid(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 400

msg_fmt = u'Unacceptable parameters.'

exception InvalidAPIVersionString(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'API Version String %(version)s is of invalid format. Must be of format MajorNum.MinorNum.'

exception **InvalidAddress**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(address)s is not a valid ip address.'

exception **InvalidAggregateAction**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

code = 400

msg_fmt = u'Unacceptable parameters.'

exception **InvalidAggregateActionAdd**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidAggregateAction

msg_fmt = u'Cannot add host to aggregate %(aggregate_id)s. Reason: %(reason)s.'

exception **InvalidAggregateActionDelete**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidAggregateAction

msg_fmt = u'Cannot remove host from aggregate %(aggregate_id)s. Reason: %(reason)s.'

exception **InvalidAggregateActionUpdate**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidAggregateAction

msg_fmt = u'Cannot update aggregate %(aggregate_id)s. Reason: %(reason)s.'

exception **InvalidAggregateActionUpdateMeta**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidAggregateAction

msg_fmt = u'Cannot update metadata of aggregate %(aggregate_id)s. Reason: %(reason)s.'

exception **InvalidArchitectureName**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Architecture name '%(arch)s' is not recognised"

exception **InvalidAssociation**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'InvalidAssociationID.NotFound'

msg_fmt = u'Invalid association.'

exception **InvalidAttribute**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Attribute not supported: %(attr)s'

exception **InvalidBDM**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Block Device Mapping is Invalid.'

exception **InvalidBDMBootSequence**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping is Invalid: Boot sequence for the instance and image/block device mapping combina

exception **InvalidBDMEphemeralSize**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Ephemeral disks requested are larger than the instance type allows.'

exception **InvalidBDMForLegacy**(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping cannot be converted to legacy format. '

exception InvalidBDMFormat(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping is Invalid: %(details)s'

exception InvalidBDMImage(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping is Invalid: failed to get image %(id)s.'

exception InvalidBDMLocalsLimit(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping is Invalid: You specified more local devices than the limit allows'

exception InvalidBDMSnapshot(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping is Invalid: failed to get snapshot %(id)s.'

exception InvalidBDMSwapSize(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Swap drive requested is larger than instance type allows.'

exception InvalidBDMVolume(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device Mapping is Invalid: failed to get volume %(id)s.'

exception InvalidBDMVolumeNotBootable(*message=None*, ***kwargs*)
Bases: nova.exception.InvalidBDM

msg_fmt = u'Block Device %(id)s is not bootable.'

exception InvalidCPUInfo(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Unacceptable CPU info: %(reason)s'

exception InvalidCidr(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(cidr)s is not a valid ip network.'

exception InvalidConnectionInfo(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid Connection Info'

exception InvalidContentType(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid content type %(content_type)s.'

exception InvalidDevicePath(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The supplied device path (%(path)s) is invalid.'

exception InvalidDiskFormat(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Disk format %(disk_format)s is not acceptable'

exception **InvalidDiskInfo**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Disk info file is invalid: %(reason)s'

exception **InvalidEc2Id**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Ec2 id %(ec2_id)s is unacceptable.'

exception **InvalidFixedIpAndMaxCountRequest**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to launch instances: %(reason)s'

exception **InvalidGlobalAPIVersion**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Version %(req_ver)s is not supported by the API. Minimum is %(min_ver)s and maximum is %(max_ver)s

exception **InvalidGroup**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Group not valid. Reason: %(reason)s'

exception **InvalidHostname**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Invalid characters in hostname '%(hostname)s"'

exception **InvalidHypervisorType**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The supplied hypervisor type of is invalid.'

exception **InvalidHypervisorVirtType**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Hypervisor virtualization type '%(hv_type)s' is not recognised"

exception **InvalidID**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid ID received %(id)s.'

exception **InvalidImageConfigDrive**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Image's config drive option '%(config_drive)s' is invalid"

exception **InvalidImageFormat**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Invalid image format '%(format)s"'

exception **InvalidImageRef**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid image href %(image_href)s.'

exception **InvalidInput**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid input received: %(reason)s'

exception **InvalidInstanceIDMalformed**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidInstanceID.Malformed'

msg_fmt = u'Invalid id: %(instance_id)s (expecting "i-...")'

exception InvalidIntValue(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(key)s must be an integer.'

exception InvalidIpAddressError(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(address)s is not a valid IP v4/6 address.'

exception InvalidIpProtocol(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid IP protocol %(protocol)s.'

exception InvalidKeypair(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidKeyPair.Format'

msg_fmt = u'Keypair data is invalid: %(reason)s'

exception InvalidLocalStorage(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'%(path)s is not on local storage: %(reason)s'

exception InvalidMetadata(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid metadata: %(reason)s'

exception InvalidMetadataSize(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid metadata size: %(reason)s'

exception InvalidParameterValue(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidParameterValue'

msg_fmt = u'%(err)s'

exception InvalidPortRange(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidParameterValue'

msg_fmt = u'Invalid port range %(from_port)s:%(to_port)s. %(msg)s'

exception InvalidQuotaMethodUsage(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Wrong quota method %(method)s used on resource %(res)s'

exception InvalidQuotaValue(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Change would make usage less than 0 for the following resources: %(unders)s'

exception InvalidRequest(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The request is invalid.'

exception InvalidReservationExpiration(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid reservation expiration %(expire)s.'

exception InvalidSharedStorage(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'%(path)s is not on shared storage: %(reason)s'

exception InvalidSortKey(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Sort key supplied was not valid.'

exception InvalidStrTime(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid datetime string: %(reason)s'

exception InvalidToken(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"The token '%(token)s' is invalid or has expired"

exception InvalidUUID(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Expected a uuid but received %(uuid)s.'

exception InvalidVLANPortGroup(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'vSwitch which contains the port group %(bridge)s is not associated with the desired physical adapter. Expe

exception InvalidVLANTag(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'VLAN tag is not appropriate for the port group %(bridge)s. Expected VLAN tag is %(tag)s, but the one as

exception InvalidVideoMode(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Provided video model (%(model)s) is not supported.'

exception InvalidVirtualMachineMode(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Virtual machine mode '%(vmmode)s' is not recognised"

exception InvalidVolume(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'UnsupportedOperation'

msg_fmt = u'Invalid volume: %(reason)s'

exception InvalidVolumeAccessMode(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid volume access mode: %(access_mode)s'

exception InvalidVolumeIDMalformed(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidVolumeID.Malformed'

msg_fmt = u'Invalid id: %(volume_id)s (expecting "i-...")'

exception InvalidWatchdogAction(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Provided watchdog action (%(action)s) is not supported.'

exception KeyManagerError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Key manager error: %(reason)s'

exception KeyPairExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

ec2_code = 'InvalidKeyPair.Duplicate'

msg_fmt = u"Key pair '%(key_name)s' already exists."

exception KeypairLimitExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Maximum number of key pairs exceeded'

exception KeypairNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'InvalidKeyPair.NotFound'

msg_fmt = u'Keypair %(name)s not found for user %(user_id)s'

exception LabelTooLong(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Maximum allowed length for 'label' is 255."

exception LiveMigrationWithOldNovaNotSafe(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Host %(server)s is running an old version of Nova, live migrations involving that version may cause data lo

exception MalformedRequestBody(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Malformed message body: %(reason)s'

exception MarkerNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Marker %(marker)s could not be found.'

exception MemoryPageSizeForbidden(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Page size %(pagesize)s forbidden against '%(against)s"'

exception MemoryPageSizeInvalid(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Invalid memory page size '%(pagesize)s"'

exception MemoryPageSizeNotSupported(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Page size %(pagesize)s is not supported by the host.'

exception **MemoryPagesUnsupported**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Host does not support guests with custom memory page sizes'

exception **MetadataLimitExceeded**(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Maximum number of metadata items exceeds %(allowed)d'

exception **MigrationError**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Migration error: %(reason)s'

exception **MigrationNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Migration %(migration_id)s could not be found.'

exception **MigrationNotFoundByStatus**(*message=None*, ***kwargs*)
Bases: nova.exception.MigrationNotFound

msg_fmt = u'Migration not found for instance %(instance_id)s with status %(status)s.'

exception **MigrationPreCheckError**(*message=None*, ***kwargs*)
Bases: nova.exception.MigrationError

msg_fmt = u'Migration pre-check error: %(reason)s'

exception **MissingParameter**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 400

ec2_code = 'MissingParameter'

msg_fmt = u'Not enough parameters: %(reason)s'

exception **MultiplePortsNotApplicable**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to launch instances: %(reason)s'

exception **NUMATopologyUnsupported**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Host does not support guests with NUMA topology set'

exception **NetworkAdapterNotFound**(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Network adapter %(adapter)s could not be found.'

exception **NetworkAmbiguous**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'More than one possible network found. Specify network ID(s) to select which one(s) to connect to.'

exception **NetworkDhcpReleaseFailed**(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to release IP %(address)s with MAC %(mac_address)s'

exception **NetworkDuplicated**(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Network %(network_id)s is duplicated.'

exception NetworkHasProject(*message=None*, ***kwargs*)
Bases: nova.exception.NetworkInUse

msg_fmt = u'Network must be disassociated from project %(project_id)s before it can be deleted.'

exception NetworkInUse(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Network %(network_id)s is still in use.'

exception NetworkMissingPhysicalNetwork(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Physical network is missing for network %(network_uuid)s'

exception NetworkNotCreated(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'%(req)s is required to create a network.'

exception NetworkNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Network %(network_id)s could not be found.'

exception NetworkNotFoundForBridge(*message=None*, ***kwargs*)
Bases: nova.exception.NetworkNotFound

msg_fmt = u'Network could not be found for bridge %(bridge)s'

exception NetworkNotFoundForCidr(*message=None*, ***kwargs*)
Bases: nova.exception.NetworkNotFound

msg_fmt = u'Network could not be found with cidr %(cidr)s.'

exception NetworkNotFoundForInstance(*message=None*, ***kwargs*)
Bases: nova.exception.NetworkNotFound

msg_fmt = u'Network could not be found for instance %(instance_id)s.'

exception NetworkNotFoundForProject(*message=None*, ***kwargs*)
Bases: nova.exception.NetworkNotFound

msg_fmt = u'Either network uuid %(network_uuid)s is not present or is not assigned to the project %(project_id)s.'

exception NetworkNotFoundForUUID(*message=None*, ***kwargs*)
Bases: nova.exception.NetworkNotFound

msg_fmt = u'Network could not be found for uuid %(uuid)s'

exception NetworkRequiresSubnet(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Network %(network_uuid)s requires a subnet in order to boot instances on.'

exception NetworkSetHostFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Network set host failed for network %(network_id)s.'

exception NoCellsAvailable(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'No cells available matching scheduling criteria.'

exception NoFixedIpsDefined(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Zero fixed ips could be found.'

exception NoFloatingIpInterface(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'UnsupportedOperation'

msg_fmt = u'Interface %(interface)s not found.'

exception NoFloatingIpsDefined(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Zero floating ips exist.'

exception NoLiveMigrationForConfigDriveInLibVirt(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Live migration of instances with config drives is not supported in libvirt unless libvirt instance path and dri

exception NoMoreFixedIps(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

ec2_code = 'UnsupportedOperation'

msg_fmt = u'No fixed IP addresses available for network: %(net)s'

exception NoMoreFloatingIps(*message=None*, ***kwargs*)
Bases: nova.exception.FloatingIpNotFound

msg_fmt = u'Zero floating ips available.'

safe = True

exception NoMoreNetworks(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'No more available networks.'

exception NoNetworksFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'No networks defined.'

exception NoUniqueMatch(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 409

msg_fmt = u'No Unique Match Found.'

exception NoValidHost(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'No valid host was found. %(reason)s'

exception NotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 404

msg_fmt = u'Resource could not be found.'

exception NovaException(*message=None*, ***kwargs*)
Bases: exceptions.Exception

Base Nova Exception

To correctly use this class, inherit from it and define a 'msg_fmt' property. That msg_fmt will get printf'd with the keyword arguments provided to the constructor.

code = 500

format_message()

headers = {}

msg_fmt = u'An unknown exception occurred.'

safe = False

exception NumaTopologyNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Instance %(instance_uuid)s does not specify a NUMA topology'

exception ObjectActionError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Object action %(action)s failed because: %(reason)s'

exception ObjectFieldInvalid(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Field %(field)s of %(objname)s is not an instance of Field'

exception OnsetFileContentLimitExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.OnsetFileLimitExceeded

msg_fmt = u'Personality file content too long'

exception OnsetFileLimitExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Personality file limit exceeded'

exception OnsetFilePathLimitExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.OnsetFileLimitExceeded

msg_fmt = u'Personality file path too long'

exception OrphanedObjectError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Cannot call %(method)s on orphaned %(objtype)s object'

exception OverQuota(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Quota exceeded for resources: %(overs)s'

exception PasteAppNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u"Could not load paste app '%(name)s' from %(path)s"

exception PciConfigInvalidWhitelist(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid PCI devices Whitelist config %(reason)s'

exception PciDeviceDetachFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to detach PCI device %(dev)s: %(reason)s'

exception PciDeviceInvalidAddressField(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Invalid PCI Whitelist: The PCI address %(address)s has an invalid %(field)s.'

exception PciDeviceInvalidDeviceName(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Invalid PCI Whitelist: The PCI whitelist can specify devname or address, but not both'

exception PciDeviceInvalidOwner(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'PCI device %(compute_node_id)s:%(address)s is owned by %(owner)s instead of %(hopeowner)s'

exception PciDeviceInvalidStatus(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'PCI device %(compute_node_id)s:%(address)s is %(status)s instead of %(hopestatus)s'

exception PciDeviceNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'PCI Device %(node_id)s:%(address)s not found.'

exception PciDeviceNotFoundById(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'PCI device %(id)s not found'

exception PciDevicePoolEmpty(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Attempt to consume PCI device %(compute_node_id)s:%(address)s from empty pool'

exception PciDevicePrepareFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to prepare PCI device %(id)s for instance %(instance_uuid)s: %(reason)s'

exception PciDeviceRequestFailed(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'PCI device request (%requests)s failed'

exception PciDeviceUnsupportedHypervisor(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'%(type)s hypervisor does not support PCI devices'

exception PciDeviceWrongAddressFormat(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'The PCI address %(address)s has an incorrect format.'

exception PciInvalidAlias(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Invalid PCI alias definition: %(reason)s'

exception PciRequestAliasNotDefined(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'PCI alias %(alias)s is not defined'

exception PluginRetriesExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Number of retries to plugin (%(num_retries)d) exceeded.'

exception PolicyNotAuthorized(*message=None*, ***kwargs*)
Bases: nova.exception.Forbidden

msg_fmt = u"Policy doesn't allow %(action)s to be performed."

exception PortInUse(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Port %(port_id)s is still in use.'

exception PortLimitExceeded(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Maximum number of ports exceeded'

exception PortNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Port id %(port_id)s could not be found.'

exception PortNotFree(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'No free port available for instance %(instance)s.'

exception PortNotUsable(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Port %(port_id)s not usable for instance %(instance)s.'

exception PortRequiresFixedIP(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Port %(port_id)s requires a FixedIP in order to be used.'

exception PreserveEphemeralNotSupported(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The current driver does not support preserving ephemeral partitions.'

exception ProjectNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Project %(project_id)s could not be found.'

exception ProjectQuotaNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaNotFound

msg_fmt = u'Quota for project %(project_id)s could not be found.'

exception ProjectUserQuotaNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaNotFound

msg_fmt = u'Quota for user %(user_id)s in project %(project_id)s could not be found.'

exception QuotaClassNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaNotFound

msg_fmt = u'Quota class %(class_name)s could not be found.'

exception `QuotaError`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

code = 413

ec2_code = 'ResourceLimitExceeded'

headers = {'Retry-After': 0}

msg_fmt = u'Quota exceeded: code=%(code)s'

safe = True

exception `QuotaExists`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Quota exists for project %(project_id)s, resource %(resource)s'

exception `QuotaNotFound`(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Quota could not be found'

exception `QuotaResourceUnknown`(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaNotFound

msg_fmt = u'Unknown quota resources %(unknown)s.'

exception `QuotaUsageNotFound`(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaNotFound

msg_fmt = u'Quota usage for project %(project_id)s could not be found.'

exception `RequestedVRamTooHigh`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'The requested amount of video memory %(req_vram)d is higher than the maximum allowed by flavor %(m

exception `RescheduledException`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Build of instance %(instance_uuid)s was re-scheduled: %(reason)s'

exception `ReservationNotFound`(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaNotFound

msg_fmt = u'Quota reservation %(uuid)s could not be found.'

exception `ResizeError`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Resize error: %(reason)s'

exception `ResourceMonitorError`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Error when creating resource monitor: %(monitor)s'

exception `RevokeCertFailure`(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Failed to revoke certificate for %(project_id)s'

exception `RngDeviceNotExist`(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'The provided RNG device path: (%(path)s) is not present on the host.'

exception RotationRequiredForBackup (*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Rotation param is required for backup image_type'

exception SchedulerHostFilterNotFound (*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Scheduler Host Filter %(filter_name)s could not be found.'

exception SecurityGroupCannotBeApplied (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Network requires port_security_enabled and subnet associated in order to apply security groups.'

exception SecurityGroupDefaultRuleNotFound (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Security group default rule (%rule_id)s not found.'

exception SecurityGroupExists (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidGroup.Duplicate'

msg_fmt = u'Security group %(security_group_name)s already exists for project %(project_id)s.'

exception SecurityGroupExistsForInstance (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Security group %(security_group_id)s is already associated with the instance %(instance_id)s'

exception SecurityGroupLimitExceeded (*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

ec2_code = 'SecurityGroupLimitExceeded'

msg_fmt = u'Maximum number of security groups or rules exceeded'

exception SecurityGroupNotExistsForInstance (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Security group %(security_group_id)s is not associated with the instance %(instance_id)s'

exception SecurityGroupNotFound (*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Security group %(security_group_id)s not found.'

exception SecurityGroupNotFoundForProject (*message=None*, ***kwargs*)
Bases: nova.exception.SecurityGroupNotFound

msg_fmt = u'Security group %(security_group_id)s not found for project %(project_id)s.'

exception SecurityGroupNotFoundForRule (*message=None*, ***kwargs*)
Bases: nova.exception.SecurityGroupNotFound

msg_fmt = u'Security group with rule %(rule_id)s not found.'

exception SecurityGroupRuleExists (*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'InvalidPermission.Duplicate'

msg_fmt = u'Rule already exists in group: %(rule)s'

exception ServiceBinaryExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Service with host %(host)s binary %(binary)s exists.'

exception ServiceGroupUnavailable(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'The service from servicegroup driver %(driver)s is temporarily unavailable.'

exception ServiceNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Service %(service_id)s could not be found.'

exception ServiceTopicExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Service with host %(host)s topic %(topic)s exists.'

exception ServiceUnavailable(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Service is unavailable at this time.'

exception ShadowTableExists(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Shadow table with name %(name)s already exists.'

exception SnapshotNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'InvalidSnapshot.NotFound'

msg_fmt = u'Snapshot %(snapshot_id)s could not be found.'

exception SocketPortInUseException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Not able to bind %(host)s:%(port)d, %(error)s'

exception SocketPortRangeExhaustedException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Not able to acquire a free port for %(host)s'

exception StorageError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Storage error: %(reason)s'

exception StorageRepositoryNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Cannot find SR to read/write VDI.'

exception SwitchNotFoundForNetworkAdapter(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Virtual switch associated with the network adapter %(adapter)s not found.'

exception TaskAlreadyRunning(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Task %(task_name)s is already running on host %(host)s'

exception TaskNotRunning(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Task %(task_name)s is not running on host %(host)s'

exception TooManyInstances(*message=None*, ***kwargs*)
Bases: nova.exception.QuotaError

msg_fmt = u'Quota exceeded for %(overs)s: Requested %(req)s, but already used %(used)d of %(allowed)d %(resource

exception UnableToMigrateToSelf(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Unable to migrate instance (%(instance_id)s) to current host (%(host)s).'

exception UnexpectedDeletingTaskStateError(*message=None*, ***kwargs*)
Bases: nova.exception.UnexpectedTaskStateError

exception UnexpectedTaskStateError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Unexpected task state: expecting %(expected)s but the actual state is %(actual)s'

exception UnexpectedVMStateError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Unexpected VM state: expecting %(expected)s but the actual state is %(actual)s'

exception UnshelveException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Error during unshelve instance %(instance_id)s: %(reason)s'

exception UnsupportedHardware(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Requested hardware '%(model)s' is not supported by the '%(virt)s' virt driver"

exception UnsupportedImageModel(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Image model '%(image)s' is not supported"

exception UnsupportedObjectError(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Unsupported object type %(objtype)s'

exception UnsupportedPolicyException(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'ServerGroup policy is not supported: %(reason)s'

exception UnsupportedVirtType(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u"Virtualization type '%(virt)s' is not supported by this compute driver"

exception ValidationError(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = '%(detail)s'

exception VersionNotFoundForAPIMethod(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'API version %(version)s is not supported on this method.'

exception VifDetailsMissingVhostuserSockPath(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'vhostuser_sock_path not present in vif_details for vif %(vif_id)s'

exception VirtualInterfaceCreateException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Virtual Interface creation failed'

exception VirtualInterfaceMacAddressException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Creation of virtual interface with unique mac address failed'

exception VirtualInterfacePlugException(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Virtual interface plugin failed'

exception VolumeBDMNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'No volume Block Device Mapping with id %(volume_id)s.'

exception VolumeBDMPathNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.VolumeBDMNotFound

msg_fmt = u'No volume Block Device Mapping at path: %(path)s'

exception VolumeDriverNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

msg_fmt = u'Could not find a handler for %(driver_type)s volume.'

exception VolumeNotCreated(*message=None*, ***kwargs*)
Bases: nova.exception.NovaException

msg_fmt = u'Volume %(volume_id)s did not finish being created even after we waited %(seconds)s seconds or %(attemp

exception VolumeNotFound(*message=None*, ***kwargs*)
Bases: nova.exception.NotFound

ec2_code = 'InvalidVolume.NotFound'

msg_fmt = u'Volume %(volume_id)s could not be found.'

exception VolumeUnattached(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

ec2_code = 'IncorrectState'

msg_fmt = u'Volume %(volume_id)s is not attached to anything'

exception VolumesNotRemoved(*message=None*, ***kwargs*)
Bases: nova.exception.Invalid

msg_fmt = u'Failed to remove volume(s): (%(reason)s)'

wrap_exception(*notifier=None*, *get_notifier=None*)
This decorator wraps a method to catch any exceptions that may get thrown. It also optionally sends the exception to the notification system.

3.7.444 The nova.filters Module

Filter support

class BaseFilter
Bases: object

Base class for all filter classes.

filter_all (*filter_obj_list*, *filter_properties*)
Yield objects that pass the filter.

Can be overridden in a subclass, if you need to base filtering decisions on all objects. Otherwise, one can just override _filter_one() to filter a single object.

run_filter_for_index (*index*)
Return True if the filter needs to be run for the "index-th" instance in a request. Only need to override this if a filter needs anything other than "first only" or "all" behaviour.

run_filter_once_per_request = False

class BaseFilterHandler (*loadable_cls_type*)
Bases: nova.loadables.BaseLoader

Base class to handle loading filter classes.

This class should be subclassed where one needs to use filters.

get_filtered_objects (*filters*, *objs*, *filter_properties*, *index=0*)

3.7.445 The nova.hacking.checks Module

class BaseASTChecker (*tree*, *filename*)
Bases: ast.NodeVisitor

Provides a simple framework for writing AST-based checks.

Subclasses should implement visit_* methods like any other AST visitor implementation. When they detect an error for a particular node the method should call `self.add_error(offending_node)`. Details about where in the code the error occurred will be pulled from the node object.

Subclasses should also provide a class variable named CHECK_DESC to be used for the human readable error message.

add_error (*node*, *message=None*)
Add an error caused by a node to the list of errors for pep8.

run ()
Called automatically by pep8.

class CheckForStrUnicodeExc (*tree*, *filename*)
Bases: nova.hacking.checks.BaseASTChecker

Checks for the use of str() or unicode() on an exception.

This currently only handles the case where str() or unicode() is used in the scope of an exception handler. If the exception is passed into a function, returned from an assertRaises, or used on an exception created in the same scope, this does not catch it.

CHECK_DESC = 'N325 str() and unicode() cannot be used on an exception. Remove or use six.text_type()'

visit_Call (*node*)

visit_TryExcept(*node*)

class CheckForTransAdd(*tree*, *filename*)
Bases: nova.hacking.checks.BaseASTChecker

Checks for the use of concatenation on a translated string.

Translations should not be concatenated with other strings, but should instead include the string being added to the translated string to give the translators the most information.

CHECK_DESC = 'N326 Translated messages cannot be concatenated. String should be included in translated message.'

TRANS_FUNC = ['_', '_LI', '_LW', '_LE', '_LC']

visit_BinOp(*node*)

assert_equal_in(*logical_line*)
Check for assertEqual(A in B, True), assertEqual(True, A in B), assertEqual(A in B, False) or assertEqual(False, A in B) sentences

N338

assert_equal_none(*logical_line*)
Check for assertEqual(A, None) or assertEqual(None, A) sentences

N318

assert_equal_type(*logical_line*)
Check for assertEqual(type(A), B) sentences

N317

assert_raises_regexp(*logical_line*)
Check for usage of deprecated assertRaisesRegexp

N335

assert_true_instance(*logical_line*)
Check for assertTrue(isinstance(a, b)) sentences

N316

assert_true_or_false_with_in(*logical_line*)
Check for assertTrue/False(A in B), assertTrue/False(A not in B), assertTrue/False(A in B, message) or assertTrue/False(A not in B, message) sentences.

N334

capital_cfg_help(*logical_line*, *tokens*)

check_api_version_decorator(*logical_line*, *previous_logical*, *blank_before*, *filename*)

check_explicit_underscore_import(*logical_line*, *filename*)
Check for explicit import of the _ function

We need to ensure that any files that are using the _() function to translate logs are explicitly importing the _ function. We can't trust unit test to catch whether the import has been added so we need to check for it here.

check_http_not_implemented(*logical_line*, *physical_line*, *filename*)

check_oslo_namespace_imports(*logical_line*, *blank_before*, *filename*)

dict_constructor_with_list_copy(*logical_line*)

factory(*register*)

import_no_db_in_virt (*logical_line*, *filename*)
Check for db calls from nova/virt

As of grizzly-2 all the database calls have been removed from nova/virt, and we want to keep it that way.

N307

import_no_virt_driver_config_deps (*physical_line*, *filename*)
Check virt drivers' config vars aren't used by other drivers

Modules under each virt driver's directory are considered private to that virt driver. Other drivers in Nova must not use their config vars. Any config vars that are to be shared should be moved into a common module

N312

import_no_virt_driver_import_deps (*physical_line*, *filename*)
Check virt drivers' modules aren't imported by other drivers

Modules under each virt driver's directory are considered private to that virt driver. Other drivers in Nova must not access those drivers. Any code that is to be shared should be refactored into a common module

N311

no_db_session_in_public_api (*logical_line*, *filename*)

no_import_translation_in_tests (*logical_line*, *filename*)
Check for 'from nova.i18n import _' N337

no_mutable_default_args (*logical_line*)

no_setting_conf_directly_in_tests (*logical_line*, *filename*)
Check for setting CONF.* attributes directly in tests

The value can leak out of tests affecting how subsequent tests run. Using self.flags(option=value) is the preferred method to temporarily set config options in tests.

N320

no_translate_debug_logs (*logical_line*, *filename*)
Check for 'LOG.debug(_('

As per our translation policy, https://wiki.openstack.org/wiki/LoggingStandards#Log_Translation we shouldn't translate debug level logs.

- This check assumes that 'LOG' is a logger.
- Use filename so we can start enforcing this in specific folders instead of needing to do so all at once.

N319

no_vi_headers (*physical_line*, *line_number*, *lines*)
Check for vi editor configuration in source files.

By default vi modelines can only appear in the first or last 5 lines of a source file.

N314

use_jsonutils (*logical_line*, *filename*)

use_timeutils_utcnow (*logical_line*, *filename*)

validate_log_translations (*logical_line*, *physical_line*, *filename*)

3.7.446 The nova.hooks Module

Decorator and config option definitions for adding custom code (hooks) around callables.

Any method may have the 'add_hook' decorator applied, which yields the ability to invoke Hook objects before or after the method. (i.e. pre and post)

Hook objects are loaded by HookLoaders. Each named hook may invoke multiple Hooks.

Example Hook object:

```
| class MyHook(object):
|    def pre(self, *args, **kwargs):
|        # do stuff before wrapped callable runs
|
|   def post(self, rv, *args, **kwargs):
|        # do stuff after wrapped callable runs
```

Example Hook object with function parameters:

```
| class MyHookWithFunction(object):
|   def pre(self, f, *args, **kwargs):
|        # do stuff with wrapped function info
|   def post(self, f, *args, **kwargs):
|        # do stuff with wrapped function info
```

exception FatalHookException

Bases: exceptions.Exception

Exception which should be raised by hooks to indicate that normal execution of the hooked function should be terminated. Raised exception will be logged and reraised.

class HookManager(*name*)

Bases: stevedore.hook.HookManager

run_post(*name*, *rv*, *args*, *kwargs*, *f=None*)

Execute optional post methods of loaded hooks.

Parameters

- **name** – The name of the loaded hooks.
- **rv** – Return values of target method call.
- **args** – Positional arguments which would be transmitted into all post methods of loaded hooks.
- **kwargs** – Keyword args which would be transmitted into all post methods of loaded hooks.
- **f** – Target function.

run_pre(*name*, *args*, *kwargs*, *f=None*)

Execute optional pre methods of loaded hooks.

Parameters

- **name** – The name of the loaded hooks.
- **args** – Positional arguments which would be transmitted into all pre methods of loaded hooks.
- **kwargs** – Keyword args which would be transmitted into all pre methods of loaded hooks.
- **f** – Target function.

add_hook(*name*, *pass_function=False*)
Execute optional pre and post methods around the decorated function. This is useful for customization around callables.

reset()
Clear loaded hooks.

3.7.447 The nova.i18n Module

oslo.i18n integration module.

See http://docs.openstack.org/developer/oslo.i18n/usage.html .

get_available_languages()

translate(*value*, *user_locale*)

3.7.448 The nova.image.api Module

Main abstraction layer for retrieving and storing information about disk images used by the compute layer.

class API
Bases: object

Responsible for exposing a relatively stable internal API for other modules in Nova to retrieve information about disk images. This API attempts to match the nova.volume.api and nova.network.api calling interface.

create(*context*, *image_info*, *data=None*)
Creates a new image record, optionally passing the image bits to backend storage.

Parameters

- **context** – The *nova.context.Context* object for the request
- **image_info** – A dict of information about the image that is passed to the image registry.
- **data** – Optional file handle or bytestream iterator that is passed to backend storage.

delete(*context*, *id_or_uri*)
Delete the information about an image and mark the image bits for deletion.

Parameters

- **context** – The *nova.context.Context* object for the request
- **id_or_uri** – A UUID identifier or an image URI to look up image information for.

download(*context*, *id_or_uri*, *data=None*, *dest_path=None*)
Transfer image bits from Glance or a known source location to the supplied destination filepath.

Parameters

- **context** – The *nova.context.RequestContext* object for the request
- **id_or_uri** – A UUID identifier or an image URI to look up image information for.
- **data** – A file object to use in downloading image data.
- **dest_path** – Filepath to transfer image bits to.

Note that because of the poor design of the *glance.ImageService.download* method, the function returns different things depending on what arguments are passed to it. If a data argument is supplied but no dest_path is specified (only done in the XenAPI virt driver's image.utils module) then None is returned from the method. If the data argument is not specified but a destination path *is* specified, then a writeable file handle to the destination path is constructed in the method and the image bits written to that file, and again, None is returned from the method. If no data argument is supplied and no dest_path argument is supplied (VMWare and XenAPI virt drivers), then the method returns an iterator to the image bits that the caller uses to write to wherever location it wants. Finally, if the allow_direct_url_schemes CONF option is set to something, then the nova.image.download modules are used to attempt to do an SCP copy of the image bits from a file location to the dest_path and None is returned after retrying one or more download locations (libvirt and Hyper-V virt drivers through nova.virt.images.fetch).

I think the above points to just how hacky/wacky all of this code is, and the reason it needs to be cleaned up and standardized across the virt driver callers.

get(*context*, *id_or_uri*, *include_locations=False*, *show_deleted=True*)

Retrieves the information record for a single disk image. If the supplied identifier parameter is a UUID, the default driver will be used to return information about the image. If the supplied identifier is a URI, then the driver that matches that URI endpoint will be used to query for image information.

Parameters

- **context** – The *nova.context.Context* object for the request
- **id_or_uri** – A UUID identifier or an image URI to look up image information for.
- **include_locations** – (Optional) include locations in the returned dict of information if the image service API supports it. If the image service API does not support the locations attribute, it will still be included in the returned dict, as an empty list.
- **show_deleted** – (Optional) show the image even the status of image is deleted.

get_all(*context*, ***kwargs*)

Retrieves all information records about all disk images available to show to the requesting user. If the requesting user is an admin, all images in an ACTIVE status are returned. If the requesting user is not an admin, the all public images and all private images that are owned by the requesting user in the ACTIVE status are returned.

Parameters

- **context** – The *nova.context.Context* object for the request
- **kwargs** – A dictionary of filter and pagination values that may be passed to the underlying image info driver.

update(*context*, *id_or_uri*, *image_info*, *data=None*, *purge_props=False*)

Update the information about an image, optionally along with a file handle or bytestream iterator for image bits. If the optional file handle for updated image bits is supplied, the image may not have already uploaded bits for the image.

Parameters

- **context** – The *nova.context.Context* object for the request
- **id_or_uri** – A UUID identifier or an image URI to look up image information for.
- **image_info** – A dict of information about the image that is passed to the image registry.
- **data** – Optional file handle or bytestream iterator that is passed to backend storage.
- **purge_props** – Optional, defaults to False. If set, the backend image registry will clear all image properties and replace them the image properties supplied in the image_info dictionary's 'properties' collection.

3.7.449 The nova.image.download.base Module

class TransferBase
Bases: object

download(*context*, *url_parts*, *dst_path*, *metadata*, ***kwargs*)

3.7.450 The nova.image.download.file Module

class FileTransfer
Bases: nova.image.download.base.TransferBase

desc_required_keys = ['id', 'mountpoint']

download(*context*, *url_parts*, *dst_file*, *metadata*, ***kwargs*)

filesystem_opts = [<oslo_config.cfg.StrOpt object at 0x7f8581006650>, <oslo_config.cfg.StrOpt object at 0x7f858100

get_download_handler(***kwargs*)

get_schemes()

3.7.451 The nova.image.glance Module

Implementation of an image service that uses Glance as the backend.

class GlanceClientWrapper(*context=None*, *host=None*, *port=None*, *use_ssl=False*, *version=1*)
Bases: object

Glance client wrapper class that implements retries.

call(*context*, *version*, *method*, **args*, ***kwargs*)
Call a glance client method. If we get a connection error, retry the request according to CONF.glance.num_retries.

class GlanceImageService(*client=None*)
Bases: object

Provides storage and retrieval of disk image objects within Glance.

create(*context*, *image_meta*, *data=None*)
Store the image data and return the new image object.

delete(*context*, *image_id*)
Delete the given image.

Raises ImageNotFound if the image does not exist.

Raises NotAuthorized if the user is not an owner.

Raises ImageNotAuthorized if the user is not authorized.

detail(*context*, ***kwargs*)
Calls out to Glance for a list of detailed image information.

download(*context*, *image_id*, *data=None*, *dst_path=None*)
Calls out to Glance for data and writes data.

show(*context*, *image_id*, *include_locations=False*, *show_deleted=True*)
Returns a dict with image data for the given opaque image id.

Parameters

- **context** – The context object to pass to image client
- **image_id** – The UUID of the image
- **include_locations** – (Optional) include locations in the returned dict of information if the image service API supports it. If the image service API does not support the locations attribute, it will still be included in the returned dict, as an empty list.
- **show_deleted** – (Optional) show the image even the status of image is deleted.

update(*context*, *image_id*, *image_meta*, *data=None*, *purge_props=True*)
Modify the given image with the new data.

class UpdateGlanceImage(*context*, *image_id*, *metadata*, *stream*)
Bases: `object`

start()

generate_glance_url()
Generate the URL to glance.

generate_identity_headers(*context*, *status='Confirmed'*)

generate_image_url(*image_ref*)
Generate an image URL from an image_ref.

get_api_servers()
Shuffle a list of CONF.glance.api_servers and return an iterator that will cycle through the list, looping around to the beginning if necessary.

get_default_image_service()

get_remote_image_service(*context*, *image_href*)
Create an image_service and parse the id from the given image_href.

The image_href param can be an href of the form 'http://example.com:9292/v1/images/b8b2c6f7-7345-4e2f-afa2-eedaba9cbbe3', or just an id such as 'b8b2c6f7-7345-4e2f-afa2-eedaba9cbbe3'. If the image_href is a standalone id, then the default image service is returned.

Parameters **image_href** – href that describes the location of an image

Returns a tuple of the form (image_service, image_id)

3.7.452 The `nova.image.s3` Module

Proxy AMI-related calls from cloud controller to objectstore service.

class S3ImageService(*service=None*, **args*, ***kwargs*)
Bases: `object`

Wraps an existing image service to support s3 based register.

create(*context*, *metadata*, *data=None*)
Create an image.

metadata['properties'] should contain image_location.

delete(*context*, *image_id*)

detail(*context*, ***kwargs*)

image_state_map = {'failed_untar': 'failed', 'available': 'available', 'downloading': 'pending', 'failed_decrypt': 'faile

show(*context*, *image_id*)

update(*context*, *image_id*, *metadata*, *data=None*)

3.7.453 The nova.ipv6.account_identifier Module

IPv6 address generation with account identifier embedded.

to_global(*prefix*, *mac*, *project_id*)

to_mac(*ipv6_address*)

3.7.454 The nova.ipv6.api Module

reset_backend()

to_global(*prefix*, *mac*, *project_id*)

to_mac(*ipv6_address*)

3.7.455 The nova.ipv6.rfc2462 Module

RFC2462 style IPv6 address generation.

to_global(*prefix*, *mac*, *project_id*)

to_mac(*ipv6_address*)

3.7.456 The nova.keymgr.barbican Module

Key manager implementation for Barbican

class BarbicanKeyManager

Bases: nova.keymgr.key_mgr.KeyManager

Key Manager Interface that wraps the Barbican client API.

copy_key(*ctxt*, *key_id*)

Copies (i.e., clones) a key stored by barbican.

Parameters

- **ctxt** – contains information of the user and the environment for the request (nova/context.py)
- **key_id** – the UUID of the key to copy

Returns the UUID of the key copy

Raises Exception if key copying fails

create_key(*ctxt*, *expiration=None*, *name='Nova Compute Key'*, *payload_content_type='application/octet-stream'*, *mode='CBC'*, *algorithm='AES'*, *length=256*)

Creates a key.

Parameters

- **ctxt** – contains information of the user and the environment for the request (nova/context.py)
- **expiration** – the date the key will expire

- **name** – a friendly name for the secret
- **payload_content_type** – the format/type of the secret data
- **mode** – the algorithm mode (e.g. CBC or CTR mode)
- **algorithm** – the algorithm associated with the secret
- **length** – the bit length of the secret

Returns the UUID of the new key

Raises Exception if key creation fails

delete_key(*ctxt*, *key_id*)
Deletes the specified key.

Parameters

- **ctxt** – contains information of the user and the environment for the request (nova/context.py)
- **key_id** – the UUID of the key to delete

Raises Exception if key deletion fails

get_key(*ctxt*, *key_id*, *payload_content_type='application/octet-stream'*)
Retrieves the specified key.

Parameters

- **ctxt** – contains information of the user and the environment for the request (nova/context.py)
- **key_id** – the UUID of the key to retrieve
- **payload_content_type** – The format/type of the secret data

Returns SymmetricKey representation of the key

Raises Exception if key retrieval fails

store_key(*ctxt*, *key*, *expiration=None*, *name='Nova Compute Key'*, *payload_content_type='application/octet-stream'*, *payload_content_encoding='base64'*, *algorithm='AES'*, *bit_length=256*, *mode='CBC'*, *from_copy=False*)
Stores (i.e., registers) a key with the key manager.

Parameters

- **ctxt** – contains information of the user and the environment for the request (nova/context.py)
- **key** – the unencrypted secret data. Known as "payload" to the barbicanclient api
- **expiration** – the expiration time of the secret in ISO 8601 format
- **name** – a friendly name for the key
- **payload_content_type** – the format/type of the secret data
- **payload_content_encoding** – the encoding of the secret data
- **algorithm** – the algorithm associated with this secret key
- **bit_length** – the bit length of this secret key
- **mode** – the algorithm mode used with this secret key

- **from_copy** – establishes whether the function is being used to copy a key. In case of the latter, it does not try to decode the key

Returns the UUID of the stored key

Raises Exception if key storage fails

3.7.457 The nova.keymgr.conf_key_mgr Module

An implementation of a key manager that reads its key from the project's configuration options.

This key manager implementation provides limited security, assuming that the key remains secret. Using the volume encryption feature as an example, encryption provides protection against a lost or stolen disk, assuming that the configuration file that contains the key is not stored on the disk. Encryption also protects the confidentiality of data as it is transmitted via iSCSI from the compute host to the storage host (again assuming that an attacker who intercepts the data does not know the secret key).

Because this implementation uses a single, fixed key, it proffers no protection once that key is compromised. In particular, different volumes encrypted with a key provided by this key manager actually share the same encryption key so *any* volume can be decrypted once the fixed key is known.

class ConfKeyManager

Bases: nova.keymgr.single_key_mgr.SingleKeyManager

This key manager implementation supports all the methods specified by the key manager interface. This implementation creates a single key in response to all invocations of create_key. Side effects (e.g., raising exceptions) for each method are handled as specified by the key manager interface.

3.7.458 The nova.keymgr.key Module

Base Key and SymmetricKey Classes

This module defines the Key and SymmetricKey classes. The Key class is the base class to represent all encryption keys. The basis for this class was copied from Java.

class Key

Bases: object

Base class to represent all keys.

get_algorithm()

Returns the key's algorithm.

Returns the key's algorithm. For example, "DSA" indicates that this key is a DSA key and "AES" indicates that this key is an AES key.

get_encoded()

Returns the key in the format specified by its encoding.

get_format()

Returns the encoding format.

Returns the key's encoding format or None if this key is not encoded.

class SymmetricKey(*alg*, *key*)

Bases: nova.keymgr.key.Key

This class represents symmetric keys.

get_algorithm()

Returns the algorithm for symmetric encryption.

get_encoded()
Returns the key in its encoded format.

get_format()
This method returns 'RAW'.

3.7.459 The nova.keymgr.key_mgr Module

Key manager API

class KeyManager
Bases: object

Base Key Manager Interface

A Key Manager is responsible for managing encryption keys for volumes. A Key Manager is responsible for creating, reading, and deleting keys.

copy_key(*ctxt*, *key_id*, ***kwargs*)
Copies (i.e., clones) a key stored by the key manager.

This method copies the specified key and returns the copy's UUID. If the specified context does not permit copying keys, then a NotAuthorized error should be raised.

Implementation note: This method should behave identically to:

```
store_key(context, get_key(context, <encryption key UUID>))
```

although it is preferable to perform this operation within the key manager to avoid unnecessary handling of the key material.

create_key(*ctxt*, *algorithm='AES'*, *length=256*, *expiration=None*, ***kwargs*)
Creates a key.

This method creates a key and returns the key's UUID. If the specified context does not permit the creation of keys, then a NotAuthorized exception should be raised.

delete_key(*ctxt*, *key_id*, ***kwargs*)
Deletes the specified key.

Implementations should verify that the caller has permission to delete the key by checking the context object (ctxt). A NotAuthorized exception should be raised if the caller lacks permission.

If the specified key does not exist, then a KeyError should be raised. Implementations should preclude users from discerning the UUIDs of keys that belong to other users by repeatedly calling this method. That is, keys that belong to other users should be considered "non- existent" and completely invisible.

get_key(*ctxt*, *key_id*, ***kwargs*)
Retrieves the specified key.

Implementations should verify that the caller has permissions to retrieve the key by checking the context object passed in as ctxt. If the user lacks permission then a NotAuthorized exception is raised.

If the specified key does not exist, then a KeyError should be raised. Implementations should preclude users from discerning the UUIDs of keys that belong to other users by repeatedly calling this method. That is, keys that belong to other users should be considered "non- existent" and completely invisible.

store_key(*ctxt*, *key*, *expiration=None*, ***kwargs*)
Stores (i.e., registers) a key with the key manager.

This method stores the specified key and returns its UUID that identifies it within the key manager. If the specified context does not permit the creation of keys, then a NotAuthorized exception should be raised.

3.7.460 The nova.keymgr.mock_key_mgr Module

A mock implementation of a key manager that stores keys in a dictionary.

This key manager implementation is primarily intended for testing. In particular, it does not store keys persistently. Lack of a centralized key store also makes this implementation unsuitable for use among different services.

Note: Instantiating this class multiple times will create separate key stores. Keys created in one instance will not be accessible from other instances of this class.

class MockKeyManager

Bases: nova.keymgr.key_mgr.KeyManager

This mock key manager implementation supports all the methods specified by the key manager interface. This implementation stores keys within a dictionary, and as a result, it is not acceptable for use across different services. Side effects (e.g., raising exceptions) for each method are handled as specified by the key manager interface.

This key manager is not suitable for use in production deployments.

copy_key(*ctxt*, *key_id*, ***kwargs*)

create_key(*ctxt*, ***kwargs*)

Creates a key.

This implementation returns a UUID for the created key. A Forbidden exception is raised if the specified context is None.

delete_key(*ctxt*, *key_id*, ***kwargs*)

Deletes the key identified by the specified id.

A Forbidden exception is raised if the context is None and a KeyError is raised if the UUID is invalid.

get_key(*ctxt*, *key_id*, ***kwargs*)

Retrieves the key identified by the specified id.

This implementation returns the key that is associated with the specified UUID. A Forbidden exception is raised if the specified context is None; a KeyError is raised if the UUID is invalid.

store_key(*ctxt*, *key*, ***kwargs*)

Stores (i.e., registers) a key with the key manager.

3.7.461 The nova.keymgr.not_implemented_key_mgr Module

Key manager implementation that raises NotImplementedError

class NotImplementedKeyManager

Bases: nova.keymgr.key_mgr.KeyManager

Key Manager Interface that raises NotImplementedError for all operations

copy_key(*ctxt*, *key_id*, ***kwargs*)

create_key(*ctxt*, *algorithm='AES'*, *length=256*, *expiration=None*, ***kwargs*)

delete_key(*ctxt*, *key_id*, ***kwargs*)

get_key(*ctxt*, *key_id*, ***kwargs*)

store_key(*ctxt*, *key*, *expiration=None*, ***kwargs*)

3.7.462 The nova.keymgr.single_key_mgr Module

An implementation of a key manager that returns a single key in response to all invocations of get_key.

class SingleKeyManager
Bases: nova.keymgr.mock_key_mgr.MockKeyManager

This key manager implementation supports all the methods specified by the key manager interface. This implementation creates a single key in response to all invocations of create_key. Side effects (e.g., raising exceptions) for each method are handled as specified by the key manager interface.

delete_key(*ctxt*, *key_id*, ***kwargs*)

store_key(*ctxt*, *key*, ***kwargs*)

3.7.463 The nova.loadables Module

Generic Loadable class support.

Meant to be used by such things as scheduler filters and weights where we want to load modules from certain directories and find certain types of classes within those modules. Note that this is quite different than generic plugins and the pluginmanager code that exists elsewhere.

Usage:

Create a directory with an __init__.py with code such as:

class SomeLoadableClass(object): pass

class MyLoader(nova.loadables.BaseLoader)

def __init__(self): super(MyLoader, self).__init__(SomeLoadableClass)

If you create modules in the same directory and subclass SomeLoadableClass within them, MyLoader().get_all_classes() will return a list of such classes.

class BaseLoader(*loadable_cls_type*)
Bases: object

get_all_classes()
Get the classes of the type we want from all modules found in the directory that defines this class.

get_matching_classes(*loadable_class_names*)
Get loadable classes from a list of names. Each name can be a full module path or the full path to a method that returns classes to use. The latter behavior is useful to specify a method that returns a list of classes to use in a default case.

3.7.464 The nova.manager Module

Base Manager class.

Managers are responsible for a certain aspect of the system. It is a logical grouping of code relating to a portion of the system. In general other components should be using the manager to make changes to the components that it is responsible for.

For example, other components that need to deal with volumes in some way, should do so by calling methods on the VolumeManager instead of directly changing fields in the database. This allows us to keep all of the code relating to volumes in the same place.

We have adopted a basic strategy of Smart managers and dumb data, which means rather than attaching methods to data objects, components should call manager methods that act on the data.

Methods on managers that can be executed locally should be called directly. If a particular method must execute on a remote host, this should be done via rpc to the service that wraps the manager

Managers should be responsible for most of the db access, and non-implementation specific data. Anything implementation specific that can't be generalized should be done by the Driver.

In general, we prefer to have one manager with multiple drivers for different implementations, but sometimes it makes sense to have multiple managers. You can think of it this way: Abstract different overall strategies at the manager level(FlatNetwork vs VlanNetwork), and different implementations at the driver level(LinuxNetDriver vs CiscoNetDriver).

Managers will often provide methods for initial setup of a host or periodic tasks to a wrapping service.

This module provides Manager, a base class for managers.

class **Manager**(*host=None*, *db_driver=None*, *service_name='undefined'*)

Bases: nova.db.base.Base, nova.openstack.common.periodic_task.PeriodicTasks

cleanup_host()

Hook to do cleanup work when the service shuts down.

Child classes should override this method.

init_host()

Hook to do additional manager initialization when one requests the service be started. This is called before any service record is created.

Child classes should override this method.

periodic_tasks(*context*, *raise_on_error=False*)

Tasks to be run at a periodic interval.

post_start_hook()

Hook to provide the manager the ability to do additional start-up work immediately after a service creates RPC consumers and starts 'running'.

Child classes should override this method.

pre_start_hook()

Hook to provide the manager the ability to do additional start-up work before any RPC queues/consumers are created. This is called after other initialization has succeeded and a service record is created.

Child classes should override this method.

3.7.465 The nova.netconf Module

3.7.466 The nova.network.api Module

class **API**(***kwargs*)

Bases: nova.network.base_api.NetworkAPI

API for doing networking via the nova-network network manager.

This is a pluggable module - other implementations do networking via other services (such as Neutron).

add_dns_entry(*context*, **args*, ***kwargs*)

Create specified DNS entry for address.

add_fixed_ip_to_instance(*context*, **args*, ***kwargs*)

Adds a fixed ip to instance from specified network.

add_network_to_project(*context*, **args*, ***kwargs*)
Force adds another network to a project.

allocate_floating_ip(*context*, **args*, ***kwargs*)
Adds (allocates) a floating ip to a project from a pool.

allocate_for_instance(*context*, **args*, ***kwargs*)
Allocates all network structures for an instance.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **vpn** – A boolean, if True, indicate a vpn to access the instance.
- **requested_networks** – A dictionary of requested_networks, Optional value containing network_id, fixed_ip, and port_id.
- **macs** – None or a set of MAC addresses that the instance should use. macs is supplied by the hypervisor driver (contrast with requested_networks which is user supplied).
- **security_groups** – None or security groups to allocate for instance.
- **dhcp_options** – None or a set of key/value pairs that should determine the DHCP BOOTP response, eg. for PXE booting an instance configured with the baremetal hypervisor. It is expected that these are already formatted for the neutron v2 api. See nova/virt/driver.py:dhcp_options_for_instance for an example.

Returns network info as from get_instance_nw_info() below

allocate_port_for_instance(*context*, *instance*, *port_id*, *network_id=None*, *requested_ip=None*)

associate(*context*, **args*, ***kwargs*)
Associate or disassociate host or project to network.

associate_floating_ip(*context*, **args*, ***kwargs*)
Associates a floating ip with a fixed ip.

Ensures floating ip is allocated to the project in context. Does not verify ownership of the fixed ip. Caller is assumed to have checked that the instance is properly owned.

cleanup_instance_network_on_host(*context*, *instance*, *host*)
Cleanup network for specified instance on host.

create(*context*, **args*, ***kwargs*)

create_pci_requests_for_sriov_ports(*context*, *pci_requests*, *requested_networks*)
Check requested networks for any SR-IOV port request.

Create a PCI request object for each SR-IOV port, and add it to the pci_requests object that contains a list of PCI request object.

create_private_dns_domain(*context*, **args*, ***kwargs*)
Create a private DNS domain with nova availability zone.

create_public_dns_domain(*context*, **args*, ***kwargs*)
Create a public DNS domain with optional nova project.

deallocate_for_instance(*context*, **args*, ***kwargs*)
Deallocates all network structures related to instance.

deallocate_port_for_instance(*context*, *instance*, *port_id*)

delete(*context*, **args*, ***kwargs*)

delete_dns_domain(*context*, **args*, ***kwargs*)
Delete the specified dns domain.

delete_dns_entry(*context*, **args*, ***kwargs*)
Delete the specified dns entry.

disassociate(*context*, **args*, ***kwargs*)

disassociate_and_release_floating_ip(*context*, *instance*, *floating_ip*)
Removes (deallocates) and deletes the floating ip.

This api call was added to allow this to be done in one operation if using neutron.

disassociate_floating_ip(*context*, **args*, ***kwargs*)
Disassociates a floating ip from fixed ip it is associated with.

get(*context*, **args*, ***kwargs*)

get_all(*context*, **args*, ***kwargs*)
Get all the networks.

If it is an admin user then api will return all the networks. If it is a normal user and nova Flat or FlatDHCP networking is being used then api will return all networks. Otherwise api will only return the networks which belong to the user's project.

get_dns_domains(*context*, **args*, ***kwargs*)
Returns a list of available dns domains. These can be used to create DNS entries for floating ips.

get_dns_entries_by_address(*context*, **args*, ***kwargs*)
Get entries for address and domain.

get_dns_entries_by_name(*context*, **args*, ***kwargs*)
Get entries for name and domain.

get_fixed_ip(*context*, **args*, ***kwargs*)

get_fixed_ip_by_address(*context*, **args*, ***kwargs*)

get_floating_ip(*context*, **args*, ***kwargs*)

get_floating_ip_by_address(*context*, **args*, ***kwargs*)

get_floating_ip_pools(*context*, **args*, ***kwargs*)

get_floating_ips_by_project(*context*, **args*, ***kwargs*)

get_instance_id_by_floating_address(*context*, **args*, ***kwargs*)

get_instance_nw_info(*context*, **args*, ***kwargs*)
Returns all network info related to an instance.

get_vif_by_mac_address(*context*, **args*, ***kwargs*)

get_vifs_by_instance(*context*, **args*, ***kwargs*)

list_ports(**args*, ***kwargs*)

migrate_instance_finish(*context*, **args*, ***kwargs*)
Finish migrating the network of an instance.

migrate_instance_start(*context*, **args*, ***kwargs*)
Start to migrate the network of an instance.

modify_dns_entry(*context*, **args*, ***kwargs*)
Create specified DNS entry for address.

release_floating_ip(*context*, **args*, ***kwargs*)
Removes (deallocates) a floating ip with address from a project.

remove_fixed_ip_from_instance(*context*, **args*, ***kwargs*)
Removes a fixed ip from instance from specified network.

setup_instance_network_on_host(*context*, *instance*, *host*)
Setup network for specified instance on host.

setup_networks_on_host(*context*, **args*, ***kwargs*)
Setup or teardown the network structures on hosts related to instance.

show_port(**args*, ***kwargs*)

validate_networks(*context*, **args*, ***kwargs*)
validate the networks passed at the time of creating the server.

Return the number of instances that can be successfully allocated with the requested network configuration.

check_policy(*context*, *action*)

wrap_check_policy(*func*)
Check policy corresponding to the wrapped methods prior to execution.

3.7.467 The nova.network.base_api Module

class NetworkAPI(*skip_policy_check=False*, ***kwargs*)
Bases: nova.db.base.Base

Base Network API for doing networking operations. New operations available on specific clients must be added here as well.

add_dns_entry(*context*, *address*, *name*, *dns_type*, *domain*)
Create specified DNS entry for address.

add_fixed_ip_to_instance(*context*, *instance*, *network_id*)
Adds a fixed ip to instance from specified network.

add_network_to_project(*context*, *project_id*, *network_uuid=None*)
Force adds another network to a project.

allocate_floating_ip(*context*, *pool=None*)
Adds (allocate) floating ip to a project from a pool.

allocate_for_instance(*context*, *instance*, *vpn*, *requested_networks*, *macs=None*, *security_groups=None*, *dhcp_options=None*)
Allocates all network structures for an instance.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **vpn** – A boolean, if True, indicate a vpn to access the instance.
- **requested_networks** – A dictionary of requested_networks, Optional value containing network_id, fixed_ip, and port_id.
- **macs** – None or a set of MAC addresses that the instance should use. macs is supplied by the hypervisor driver (contrast with requested_networks which is user supplied).
- **security_groups** – None or security groups to allocate for instance.

- **dhcp_options** – None or a set of key/value pairs that should determine the DHCP BOOTP response, eg. for PXE booting an instance configured with the baremetal hypervisor. It is expected that these are already formatted for the neutron v2 api. See nova/virt/driver.py:dhcp_options_for_instance for an example.

Returns network info as from get_instance_nw_info() below

allocate_port_for_instance(*context*, *instance*, *port_id*, *network_id=None*, *requested_ip=None*)
Allocate port for instance.

associate(*context*, *network_uuid*, *host=<object object at 0x7f858d3efe20>*, *project=<object object at 0x7f858d3efe20>*)
Associate or disassociate host or project to network.

associate_floating_ip(*context*, *instance*, *floating_address*, *fixed_address*, *affect_auto_assigned=False*)
Associates a floating ip with a fixed ip.

cleanup_instance_network_on_host(*context*, *instance*, *host*)
Cleanup network for specified instance on host.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **host** – The host which network should be cleanup for instance.

create(*context*, ***kwargs*)
Create a network.

create_pci_requests_for_sriov_ports(*context*, *pci_requests*, *requested_networks*)
Check requested networks for any SR-IOV port request.

Create a PCI request object for each SR-IOV port, and add it to the pci_requests object that contains a list of PCI request object.

create_private_dns_domain(*context*, *domain*, *availability_zone*)
Create a private DNS domain with nova availability zone.

create_public_dns_domain(*context*, *domain*, *project=None*)
Create a public DNS domain with optional nova project.

deallocate_for_instance(*context*, *instance*, *requested_networks=None*)
Deallocates all network structures related to instance.

deallocate_port_for_instance(*context*, *instance*, *port_id*)
Deallocate port for instance.

delete(*context*, *network_uuid*)
Delete a specific network.

delete_dns_domain(*context*, *domain*)
Delete the specified dns domain.

delete_dns_entry(*context*, *name*, *domain*)
Delete the specified dns entry.

disassociate(*context*, *network_uuid*)
Disassociate a network for client.

disassociate_and_release_floating_ip(*context*, *instance*, *floating_ip*)
Removes (deallocates) and deletes the floating ip.

disassociate_floating_ip(*context*, *instance*, *address*, *affect_auto_assigned=False*)
Disassociates a floating ip from fixed ip it is associated with.

get(*context*, *network_uuid*)
Get specific network for client.

get_all(*context*)
Get all the networks for client.

get_dns_domains(*context*)
Returns a list of available dns domains. These can be used to create DNS entries for floating ips.

get_dns_entries_by_address(*context*, *address*, *domain*)
Get entries for address and domain.

get_dns_entries_by_name(*context*, *name*, *domain*)
Get entries for name and domain.

get_fixed_ip(*context*, *id*)
Get fixed ip by id.

get_fixed_ip_by_address(*context*, *address*)
Get fixed ip by address.

get_floating_ip(*context*, *id*)
Get floating ip by id.

get_floating_ip_by_address(*context*, *address*)
Get floating ip by address.

get_floating_ip_pools(*context*)
Get floating ip pools.

get_floating_ips_by_project(*context*)
Get floating ips by project.

get_instance_id_by_floating_address(*context*, *address*)
Get instance id by floating address.

get_instance_nw_info(*context*, *instance*, ***kwargs*)
Returns all network info related to an instance.

get_vif_by_mac_address(*context*, *mac_address*)
Get vif mac address.

get_vifs_by_instance(*context*, *instance*)
Get vifs by instance.

list_ports(**args*, ***kwargs*)
List ports.

migrate_instance_finish(*context*, *instance*, *migration*)
Finish migrating the network of an instance.

migrate_instance_start(*context*, *instance*, *migration*)
Start to migrate the network of an instance.

modify_dns_entry(*context*, *name*, *address*, *domain*)
Create specified DNS entry for address.

release_floating_ip(*context*, *address*, *affect_auto_assigned=False*)
Removes (deallocates) a floating ip with address from a project.

remove_fixed_ip_from_instance(*context*, *instance*, *address*)
Removes a fixed ip from instance from specified network.

setup_instance_network_on_host(*context*, *instance*, *host*)
Setup network for specified instance on host.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **host** – The host which network should be setup for instance.

setup_networks_on_host(*context*, *instance*, *host=None*, *teardown=False*)
Setup or teardown the network structures on hosts related to instance.

show_port(**args*, ***kwargs*)
Show specific port.

validate_networks(*context*, *requested_networks*, *num_instances*)
validate the networks passed at the time of creating the server.

Return the number of instances that can be successfully allocated with the requested network configuration.

refresh_cache(*f*)
Decorator to update the instance_info_cache

Requires context and instance as function args

update_instance_cache_with_nw_info(**args*, ***kwargs*)

3.7.468 The nova.network.dns_driver Module

class DNSDriver
Bases: object

Defines the DNS manager interface. Does nothing.

create_domain(*_fqdomain*)

create_entry(*_name*, *_address*, *_type*, *_domain*)

delete_domain(*_fqdomain*)

delete_entry(*_name*, *_domain*)

get_domains()

get_entries_by_address(*_address*, *_domain*)

get_entries_by_name(*_name*, *_domain*)

modify_address(*_name*, *_address*, *_domain*)

3.7.469 The nova.network.driver Module

load_network_driver(*network_driver=None*)

3.7.470 The nova.network.floating_ips Module

class FloatingIP
Bases: object

Mixin class for adding floating IP functionality to a manager.

add_dns_entry(*context*, *address*, *name*, *dns_type*, *domain*)

allocate_floating_ip(*context*, *project_id*, *auto_assigned=False*, *pool=None*)
Gets a floating ip from the pool.

allocate_for_instance(*context*, ***kwargs*)
Handles allocating the floating IP resources for an instance.

calls super class allocate_for_instance() as well

rpc.called by network_api

associate_floating_ip(**args*, ***kwargs*)

create_private_dns_domain(*context*, *domain*, *av_zone*)

create_public_dns_domain(*context*, *domain*, *project*)

deallocate_floating_ip(**args*, ***kwargs*)

deallocate_for_instance(*context*, ***kwargs*)
Handles deallocating floating IP resources for an instance.

calls super class deallocate_for_instance() as well.

rpc.called by network_api

delete_dns_domain(*context*, *domain*)

delete_dns_entry(*context*, *name*, *domain*)

disassociate_floating_ip(**args*, ***kwargs*)

get_dns_domains(*context*)

get_dns_entries_by_address(*context*, *address*, *domain*)

get_dns_entries_by_name(*context*, *name*, *domain*)

get_floating_ip(**args*, ***kwargs*)

get_floating_ip_by_address(*context*, *address*)
Returns a floating IP as a dict.

get_floating_ip_pools(*context*)
Returns list of floating ip pools.

get_floating_ips_by_fixed_address(*context*, *fixed_address*)
Returns the floating IPs associated with a fixed_address.

get_floating_ips_by_project(*context*)
Returns the floating IPs allocated to a project.

get_floating_pools(*context*)
Returns list of floating pools.

init_host_floating_ips()
Configures floating ips owned by host.

migrate_instance_finish(*context*, *instance_uuid*, *floating_addresses*, *host=None*, *rxtx_factor=None*, *project_id=None*, *source=None*, *dest=None*)

migrate_instance_start (*context*, *instance_uuid*, *floating_addresses*, *rxtx_factor=None*, *project_id=None*, *source=None*, *dest=None*)

modify_dns_entry (*context*, *address*, *name*, *domain*)

servicegroup_api = **None**

class LocalManager

Bases: nova.db.base.Base, nova.network.floating_ips.FloatingIP

3.7.471 The nova.network.l3 Module

class L3Driver (*l3_lib=None*)

Bases: object

Abstract class that defines a generic L3 API.

add_floating_ip (*floating_ip*, *fixed_ip*, *l3_interface_id*, *network=None*)

Add a floating IP bound to the fixed IP with an optional l3_interface_id. Some drivers won't care about the l3_interface_id so just pass None in that case. Network is also an optional parameter.

add_vpn (*public_ip*, *port*, *private_ip*)

clean_conntrack (*fixed_ip*)

initialize (***kwargs*)

Set up basic L3 networking functionality.

initialize_gateway (*network*)

Set up a gateway on this network.

initialize_network (*network*)

Enable rules for a specific network.

is_initialized()

Returns True/False (whether the driver is initialized).

remove_floating_ip (*floating_ip*, *fixed_ip*, *l3_interface_id*, *network=None*)

remove_gateway (*network_ref*)

Remove an existing gateway on this network.

remove_vpn (*public_ip*, *port*, *private_ip*)

teardown()

class LinuxNetL3

Bases: nova.network.l3.L3Driver

L3 driver that uses linux_net as the backend.

add_floating_ip (*floating_ip*, *fixed_ip*, *l3_interface_id*, *network=None*)

add_vpn (*public_ip*, *port*, *private_ip*)

initialize (***kwargs*)

initialize_gateway (*network_ref*)

initialize_network (*cidr*, *is_external*)

is_initialized()

remove_floating_ip (*floating_ip*, *fixed_ip*, *l3_interface_id*, *network=None*)

remove_gateway(*network_ref*)

remove_vpn(*public_ip*, *port*, *private_ip*)

teardown()

class **NullL3**
Bases: nova.network.l3.L3Driver

The L3 driver that doesn't do anything. This class can be used when nova-network should not manipulate L3 forwarding at all (e.g., in a Flat or FlatDHCP scenario).

add_floating_ip(*floating_ip*, *fixed_ip*, *l3_interface_id*, *network=None*)

add_vpn(*public_ip*, *port*, *private_ip*)

clean_conntrack(*fixed_ip*)

initialize(***kwargs*)

initialize_gateway(*network_ref*)

initialize_network(*cidr*)

is_initialized()

remove_floating_ip(*floating_ip*, *fixed_ip*, *l3_interface_id*, *network=None*)

remove_gateway(*network_ref*)

remove_vpn(*public_ip*, *port*, *private_ip*)

teardown()

3.7.472 The nova.network.ldapdns Module

class **DNSEntry**(*ldap_object*)
Bases: object

dn

rdn

class **DomainEntry**(*ldap_object*, *domain*)
Bases: nova.network.ldapdns.DNSEntry

add_entry(*name*, *address*)

classmethod **create_domain**(*lobj*, *domain*)
Create a new domain entry, and return an object that wraps it.

delete()
Delete the domain that this entry refers to.

remove_entry(*name*)

subentries_with_ip(*ip*)

subentry_with_name(*name*)

update_soa()

class **HostEntry**(*parent*, *tuple*)
Bases: nova.network.ldapdns.DNSEntry

ip

modify_address(*name*, *address*)

names

parent

remove_name(*name*)

class LdapDNS

Bases: nova.network.dns_driver.DNSDriver

Driver for PowerDNS using ldap as a back end.

This driver assumes ldap-method=strict, with all domains in the top-level, aRecords only.

create_domain(*domain*)

create_entry(*name*, *address*, *type*, *domain*)

delete_dns_file()

delete_domain(*domain*)

delete_entry(*name*, *domain*)

get_domains()

get_entries_by_address(*address*, *domain*)

get_entries_by_name(*name*, *domain*)

modify_address(*name*, *address*, *domain*)

create_modlist(*newattrs*)

3.7.473 The nova.network.linux_net Module

Implements vlans, bridges, and iptables rules using linux utilities.

class IptablesManager(*execute=None*)

Bases: object

Wrapper for iptables.

See IptablesTable for some usage docs

A number of chains are set up to begin with.

First, nova-filter-top. It's added at the top of FORWARD and OUTPUT. Its name is not wrapped, so it's shared between the various nova workers. It's intended for rules that need to live at the top of the FORWARD and OUTPUT chains. It's in both the ipv4 and ipv6 set of tables.

For ipv4 and ipv6, the built-in INPUT, OUTPUT, and FORWARD filter chains are wrapped, meaning that the "real" INPUT chain has a rule that jumps to the wrapped INPUT chain, etc. Additionally, there's a wrapped chain named "local" which is jumped to from nova-filter-top.

For ipv4, the built-in PREROUTING, OUTPUT, and POSTROUTING nat chains are wrapped in the same was as the built-in filter chains. Additionally, there's a snat chain that is applied after the POSTROUTING chain.

apply()

defer_apply_off()

defer_apply_on()

dirty()

class IptablesRule(*chain*, *rule*, *wrap=True*, *top=False*)
Bases: object

An iptables rule.

You shouldn't need to use this class directly, it's only used by IptablesManager.

class IptablesTable
Bases: object

An iptables table.

add_chain(*name*, *wrap=True*)
Adds a named chain to the table.

The chain name is wrapped to be unique for the component creating it, so different components of Nova can safely create identically named chains without interfering with one another.

At the moment, its wrapped name is <binary name>-<chain name>, so if nova-compute creates a chain named 'OUTPUT', it'll actually end up named 'nova-compute-OUTPUT'.

add_rule(*chain*, *rule*, *wrap=True*, *top=False*)
Add a rule to the table.

This is just like what you'd feed to iptables, just without the '-A <chain name>' bit at the start.

However, if you need to jump to one of your wrapped chains, prepend its name with a '$' which will ensure the wrapping is applied correctly.

empty_chain(*chain*, *wrap=True*)
Remove all rules from a chain.

has_chain(*name*, *wrap=True*)

remove_chain(*name*, *wrap=True*)
Remove named chain.

This removal "cascades". All rule in the chain are removed, as are all rules in other chains that jump to it.

If the chain is not found, this is merely logged.

remove_rule(*chain*, *rule*, *wrap=True*, *top=False*)
Remove a rule from a chain.

Note: The rule must be exactly identical to the one that was added. You cannot switch arguments around like you can with the iptables CLI tool.

remove_rules_regex(*regex*)
Remove all rules matching regex.

class LinuxBridgeInterfaceDriver
Bases: nova.network.linux_net.LinuxNetInterfaceDriver

static ensure_bridge(**args*, ***kwargs*)
Create a bridge unless it already exists.

Parameters

- **interface** – the interface to create the bridge on.
- **net_attrs** – dictionary with attributes used to create bridge.
- **gateway** – whether or not the bridge is a gateway.
- **filtering** – whether or not to create filters on the bridge.

If net_attrs is set, it will add the net_attrs['gateway'] to the bridge using net_attrs['broadcast'] and net_attrs['cidr']. It will also add the ip_v6 address specified in net_attrs['cidr_v6'] if use_ipv6 is set.

The code will attempt to move any ips that already exist on the interface onto the bridge and reset the default gateway if necessary.

static ensure_vlan (**args*, ***kwargs*)
Create a vlan unless it already exists.

static ensure_vlan_bridge (*vlan_num*, *bridge*, *bridge_interface*, *net_attrs=None*, *mac_address=None*, *mtu=None*)
Create a vlan and bridge unless they already exist.

get_dev (*network*)

plug (*network*, *mac_address*, *gateway=True*)

static remove_bridge (**args*, ***kwargs*)
Delete a bridge.

static remove_vlan (**args*, ***kwargs*)
Delete a vlan.

static remove_vlan_bridge (*vlan_num*, *bridge*)
Delete a bridge and vlan.

unplug (*network*, *gateway=True*)

class LinuxNetInterfaceDriver
Bases: `object`

Abstract class that defines generic network host API for all Linux interface drivers.

get_dev (*network*)
Get device name.

plug (*network*, *mac_address*)
Create Linux device, return device name.

unplug (*network*)
Destroy Linux device, return device name.

class LinuxOVSInterfaceDriver
Bases: `nova.network.linux_net.LinuxNetInterfaceDriver`

get_dev (*network*)

plug (*network*, *mac_address*, *gateway=True*)

unplug (*network*)

class NeutronLinuxBridgeInterfaceDriver
Bases: `nova.network.linux_net.LinuxNetInterfaceDriver`

BRIDGE_NAME_PREFIX = 'brq'

GATEWAY_INTERFACE_PREFIX = 'gw-'

get_bridge (*network*)

get_dev (*network*)

plug (*network*, *mac_address*, *gateway=True*)

unplug (*network*)

QuantumLinuxBridgeInterfaceDriver
alias of NeutronLinuxBridgeInterfaceDriver

add_snat_rule(*ip_range*, *is_external=False*)

bind_floating_ip(*floating_ip*, *device*)
Bind ip to public interface.

clean_conntrack(*fixed_ip*)

create_ivs_vif_port(*dev*, *iface_id*, *mac*, *instance_id*)

create_ovs_vif_port(*bridge*, *dev*, *iface_id*, *mac*, *instance_id*)

create_tap_dev(*dev*, *mac_address=None*)

delete_bridge_dev(*dev*)
Delete a network bridge.

delete_ivs_vif_port(*dev*)

delete_net_dev(*dev*)
Delete a network device only if it exists.

delete_ovs_vif_port(*bridge*, *dev*)

device_exists(*device*)
Check if ethernet device exists.

ensure_ebtables_rules(**args*, ***kwargs*)

ensure_floating_forward(*floating_ip*, *fixed_ip*, *device*, *network*)
Ensure floating ip forwarding rule.

ensure_metadata_ip()
Sets up local metadata ip.

ensure_vpn_forward(*public_ip*, *port*, *private_ip*)
Sets up forwarding rules for vlan.

floating_ebtables_rules(*fixed_ip*, *network*)
Makes sure only in-network traffic is bridged.

floating_forward_rules(*floating_ip*, *fixed_ip*, *device*)

get_binary_name()
Grab the name of the binary we're running in.

get_dev(*network*)

get_dhcp_hosts(*context*, *network_ref*, *fixedips*)
Get network's hosts config in dhcp-host format.

get_dhcp_leases(*context*, *network_ref*)
Return a network's hosts config in dnsmasq leasefile format.

get_dhcp_opts(*context*, *network_ref*, *fixedips*)
Get network's hosts config in dhcp-opts format.

get_dns_hosts(*context*, *network_ref*)
Get network's DNS hosts in hosts format.

get_gateway_rules(*bridge*)

init_host(*ip_range*, *is_external=False*)
Basic networking setup goes here.

initialize_gateway_device(**args*, ***kwargs*)

is_pid_cmdline_correct(*pid*, *match*)
Ensure that the cmdline for a pid seems sane

Because pids are recycled, blindly killing by pid is something to avoid. This provides the ability to include a substring that is expected in the cmdline as a safety check.

isolate_dhcp_address(*interface*, *address*)

kill_dhcp(*dev*)

metadata_accept()
Create the filter accept rule for metadata.

metadata_forward()
Create forwarding rule for metadata.

ovs_set_vhostuser_port_type(*dev*)

plug(*network*, *mac_address*, *gateway=True*)

release_dhcp(*dev*, *address*, *mac_address*)

remove_ebtables_rules(**args*, ***kwargs*)

remove_floating_forward(*floating_ip*, *fixed_ip*, *device*, *network*)
Remove forwarding for floating ip.

remove_isolate_dhcp_address(*interface*, *address*)

restart_dhcp(**args*, ***kwargs*)
(Re)starts a dnsmasq server for a given network.

If a dnsmasq instance is already running then send a HUP signal causing it to reload, otherwise spawn a new instance.

send_arp_for_ip(*ip*, *device*, *count*)

set_vf_interface_vlan(*pci_addr*, *mac_addr*, *vlan=0*)

unbind_floating_ip(*floating_ip*, *device*)
Unbind a public ip from public interface.

unplug(*network*)

update_dhcp(*context*, *dev*, *network_ref*)

update_dhcp_hostfile_with_text(*dev*, *hosts_text*)

update_dns(*context*, *dev*, *network_ref*)

update_ra(**args*, ***kwargs*)

write_to_file(*file*, *data*, *mode='w'*)

3.7.474 The `nova.network.manager` Module

Network Hosts are responsible for allocating ips and setting up network.

There are multiple backend drivers that handle specific types of networking topologies. All of the network commands are issued to a subclass of `NetworkManager`.

class `FlatDHCPManager`(*network_driver=None*, **args*, ***kwargs*)

Bases: `nova.network.manager.RPCAllocateFixedIP`, `nova.network.floating_ips.FloatingIP`, `nova.network.manager.NetworkManager`

Flat networking with dhcp.

FlatDHCPManager will start up one dhcp server to give out addresses. It never injects network settings into the guest. It also manages bridges. Otherwise it behaves like FlatManager.

`DHCP` = True

`SHOULD_CREATE_BRIDGE` = True

`init_host`()

Do any initialization that needs to be run if this is a standalone service.

`required_create_args` = ['bridge']

class `FlatManager`(*network_driver=None*, **args*, ***kwargs*)

Bases: `nova.network.manager.NetworkManager`

Basic network where no vlans are used.

FlatManager does not do any bridge or vlan creation. The user is responsible for setting up whatever bridges are specified when creating networks through nova-manage. This bridge needs to be created on all compute hosts.

The idea is to create a single network for the host with a command like: nova-manage network create 192.168.0.0/24 1 256. Creating multiple networks for one manager is currently not supported, but could be added by modifying allocate_fixed_ip and get_network to get the network with new logic. Arbitrary lists of addresses in a single network can be accomplished with manual db editing.

If flat_injected is True, the compute host will attempt to inject network config into the guest. It attempts to modify /etc/network/interfaces and currently only works on debian based systems. To support a wider range of OSes, some other method may need to be devised to let the guest know which ip it should be using so that it can configure itself. Perhaps an attached disk or serial device with configuration info.

Metadata forwarding must be handled by the gateway, and since nova does not do any setup in this mode, it must be done manually. Requests to 169.254.169.254 port 80 will need to be forwarded to the api server.

`allocate_floating_ip`(*context*, *project_id*, *pool*)

Gets a floating ip from the pool.

`associate_floating_ip`(*context*, *floating_address*, *fixed_address*, *affect_auto_assigned=False*)

Associates a floating ip with a fixed ip.

Makes sure everything makes sense then calls _associate_floating_ip, rpc'ing to correct host if i'm not it.

`deallocate_fixed_ip`(*context*, *address*, *host=None*, *teardown=True*, *instance=None*)

Returns a fixed ip to the pool.

`deallocate_floating_ip`(*context*, *address*, *affect_auto_assigned*)

Returns a floating ip to the pool.

`disassociate_floating_ip`(*context*, *address*, *affect_auto_assigned=False*)

Disassociates a floating ip from its fixed ip.

Makes sure everything makes sense then calls _disassociate_floating_ip, rpc'ing to correct host if i'm not it.

`get_floating_ip`(*context*, *id*)

Returns a floating IP as a dict.

`get_floating_ip_by_address`(*context*, *address*)

Returns a floating IP as a dict.

get_floating_ip_pools(*context*)
Returns list of floating ip pools.

get_floating_ips_by_fixed_address(*context*, *fixed_address*)
Returns the floating IPs associated with a fixed_address.

get_floating_ips_by_project(*context*)
Returns the floating IPs allocated to a project.

get_floating_pools(*context*)
Returns list of floating pools.

migrate_instance_finish(*context*, *instance_uuid*, *floating_addresses*, *host=None*, *rxtx_factor=None*, *project_id=None*, *source=None*, *dest=None*)

migrate_instance_start(*context*, *instance_uuid*, *floating_addresses*, *rxtx_factor=None*, *project_id=None*, *source=None*, *dest=None*)

required_create_args = ['bridge']

timeout_fixed_ips = False

update_dns(*context*, *network_ids*)
Called when fixed IP is allocated or deallocated.

class **NetworkManager**(*network_driver=None*, **args*, ***kwargs*)
Bases: nova.manager.Manager

Implements common network manager functionality.

This class must be subclassed to support specific topologies.

host management: hosts configure themselves for networks they are assigned to in the table upon startup. If there are networks in the table which do not have hosts, those will be filled in and have hosts configured as the hosts pick them up one at time during their periodic task. The one at a time part is to flatten the layout to help scale

DHCP = False

SHOULD_CREATE_BRIDGE = False

SHOULD_CREATE_VLAN = False

add_fixed_ip_to_instance(*context*, *instance_id*, *host*, *network_id*, *rxtx_factor=None*)
Adds a fixed ip to an instance from specified network.

add_network_to_project(*ctxt*, *project_id*, *network_uuid*)

allocate_fixed_ip(*context*, *instance_id*, *network*, ***kwargs*)
Gets a fixed ip from the pool.

allocate_for_instance(*context*, ***kwargs*)
Handles allocating the various network resources for an instance.

rpc.called by network_api

create_networks(*context*, *label*, *cidr=None*, *multi_host=None*, *num_networks=None*, *network_size=None*, *cidr_v6=None*, *gateway=None*, *gateway_v6=None*, *bridge=None*, *bridge_interface=None*, *dns1=None*, *dns2=None*, *fixed_cidr=None*, *allowed_start=None*, *allowed_end=None*, ***kwargs*)

deallocate_fixed_ip(*context*, *address*, *host=None*, *teardown=True*, *instance=None*)
Returns a fixed ip to the pool.

deallocate_for_instance (*context*, ***kwargs*)
Handles deallocating various network resources for an instance.

rpc.called by network_api kwargs can contain fixed_ips to circumvent another db lookup

delete_network (*context*, *fixed_range*, *uuid*, *require_disassociated=True*)

disassociate_network (*context*, *network_uuid*)

get_all_networks (*context*)

get_backdoor_port (*context*)
Return backdoor port for eventlet_backdoor.

get_dhcp_leases (*ctxt*, *network_ref*)
Broker the request to the driver to fetch the dhcp leases.

get_fixed_ip (*context*, *id*)
Return a fixed ip.

get_fixed_ip_by_address (*context*, *address*)

get_instance_id_by_floating_address (*context*, *address*)
Returns the instance id a floating ip's fixed ip is allocated to.

get_instance_nw_info (**args*, ***kwargs*)

get_instance_uuids_by_ip_filter (*context*, *filters*)

get_network (*context*, *network_uuid*)

get_vif_by_mac_address (*context*, *mac_address*)
Returns the vifs record for the mac_address.

get_vifs_by_instance (*context*, *instance_id*)
Returns the vifs associated with an instance.

init_host ()
Do any initialization that needs to be run if this is a standalone service.

lease_fixed_ip (*context*, *address*)
Called by dhcp-bridge when ip is leased.

release_fixed_ip (*context*, *address*, *mac=None*)
Called by dhcp-bridge when ip is released.

remove_fixed_ip_from_instance (*context*, *instance_id*, *host*, *address*, *rxtx_factor=None*)
Removes a fixed ip from an instance from specified network.

required_create_args = []

rpc_setup_network_on_host (*context*, *network_id*, *teardown*)

set_network_host (*context*, *network_ref*)
Safely sets the host of the network.

setup_networks_on_host (*context*, *instance_id*, *host*, *teardown=False*)
calls setup/teardown on network hosts for an instance.

target = <Target version=1.14>

timeout_fixed_ips = True

update_dns (*context*, *network_ids*)
Called when fixed IP is allocated or deallocated.

validate_networks (*context*, *networks*)
check if the networks exists and host is set to each network.

class RPCAllocateFixedIP
Bases: object

Mixin class originally for FlatDCHP and VLAN network managers.

used since they share code to RPC.call allocate_fixed_ip on the correct network host to configure dnsmasq

deallocate_fixed_ip (*context*, *address*, *host=None*, *teardown=True*, *instance=None*)
Call the superclass deallocate_fixed_ip if i'm the correct host otherwise call to the correct host

servicegroup_api = None

class VlanManager (*network_driver=None*, **args*, ***kwargs*)
Bases: nova.network.manager.RPCAllocateFixedIP, nova.network.floating_ips.FloatingIP, nova.network.manager.NetworkManager

Vlan network with dhcp.

VlanManager is the most complicated. It will create a host-managed vlan for each project. Each project gets its own subnet. The networks and associated subnets are created with nova-manage using a command like: nova-manage network create 10.0.0.0/8 3 16. This will create 3 networks of 16 addresses from the beginning of the 10.0.0.0 range.

A dhcp server is run for each subnet, so each project will have its own. For this mode to be useful, each project will need a vpn to access the instances in its subnet.

DHCP = True

SHOULD_CREATE_BRIDGE = True

SHOULD_CREATE_VLAN = True

add_network_to_project (*context*, *project_id*, *network_uuid=None*)
Force adds another network to a project.

allocate_fixed_ip (*context*, *instance_id*, *network*, ***kwargs*)
Gets a fixed ip from the pool.

associate (*context*, *network_uuid*, *associations*)
Associate or disassociate host or project to network.

create_networks (*context*, ***kwargs*)
Create networks based on parameters.

init_host ()
Do any initialization that needs to be run if this is a standalone service.

required_create_args = ['bridge_interface']

3.7.475 The nova.network.minidns Module

class MiniDNS
Bases: nova.network.dns_driver.DNSDriver

Trivial DNS driver. This will read/write to a local, flat file and have no effect on your actual DNS system. This class is strictly for testing purposes, and should keep you out of dependency hell.

Note that there is almost certainly a race condition here that will manifest anytime instances are rapidly created and deleted. A proper implementation will need some manner of locking.

create_domain (*fqdomain*)

create_entry(*name*, *address*, *type*, *domain*)

delete_dns_file()

delete_domain(*fqdomain*)

delete_entry(*name*, *domain*)

get_domains()

get_entries_by_address(*address*, *domain*)

get_entries_by_name(*name*, *domain*)

modify_address(*name*, *address*, *domain*)

parse_line(*line*)

qualify(*name*, *domain*)

3.7.476 The nova.network.model Module

class FixedIP(*floating_ips=None*, ***kwargs*)
Bases: nova.network.model.IP

Represents a Fixed IP address in Nova.

add_floating_ip(*floating_ip*)

floating_ip_addresses()

static hydrate(*fixed_ip*)

class IP(*address=None*, *type=None*, ***kwargs*)
Bases: nova.network.model.Model

Represents an IP address in Nova.

classmethod hydrate(*ip*)

is_in_subnet(*subnet*)

class Model
Bases: dict

Defines some necessary structures for most of the network models.

get_meta(*key*, *default=None*)
calls get(key, default) on self['meta'].

class Network(*id=None*, *bridge=None*, *label=None*, *subnets=None*, ***kwargs*)
Bases: nova.network.model.Model

Represents a Network in Nova.

add_subnet(*subnet*)

classmethod hydrate(*network*)

class NetworkInfo
Bases: list

Stores and manipulates network information for a Nova instance.

fixed_ips()
Returns all fixed_ips without floating_ips attached.

floating_ips()
Returns all floating_ips.

classmethod hydrate(*network_info*)

json()

wait(*do_raise=True*)
A no-op method.

This is useful to avoid type checking when NetworkInfo might be subclassed with NetworkInfoAsyncWrapper.

class NetworkInfoAsyncWrapper(*async_method*, **args*, ***kwargs*)
Bases: nova.network.model.NetworkInfo

Wrapper around NetworkInfo that allows retrieving NetworkInfo in an async manner.

This allows one to start querying for network information before you know you will need it. If you have a long-running operation, this allows the network model retrieval to occur in the background. When you need the data, it will ensure the async operation has completed.

As an example:

def allocate_net_info(arg1, arg2) return call_neutron_to_allocate(arg1, arg2)

network_info = NetworkInfoAsyncWrapper(allocate_net_info, arg1, arg2) [do a long running operation – real network_info will be retrieved in the background] [do something with network_info]

wait(*do_raise=True*)
Wait for async call to finish.

class Route(*cidr=None*, *gateway=None*, *interface=None*, ***kwargs*)
Bases: nova.network.model.Model

Represents an IP Route in Nova.

classmethod hydrate(*route*)

class Subnet(*cidr=None*, *dns=None*, *gateway=None*, *ips=None*, *routes=None*, ***kwargs*)
Bases: nova.network.model.Model

Represents a Subnet in Nova.

add_dns(*dns*)

add_ip(*ip*)

add_route(*new_route*)

as_netaddr()
Convience function to get cidr as a netaddr object.

classmethod hydrate(*subnet*)

class VIF(*id=None*, *address=None*, *network=None*, *type=None*, *details=None*, *devname=None*, *ovs_interfaceid=None*, *qbh_params=None*, *qbg_params=None*, *active=False*, *vnic_type='normal'*, *profile=None*, *preserve_on_delete=False*, ***kwargs*)
Bases: nova.network.model.Model

Represents a Virtual Interface in Nova.

fixed_ips()

floating_ips()

get_physical_network()

classmethod hydrate(*vif*)

is_hybrid_plug_enabled()

is_neutron_filtering_enabled()

labeled_ips()
Returns the list of all IPs

The return value looks like this flat structure:

```
{'network_label': 'my_network',
 'network_id': 'n8v29837fn234782f08fjxk3ofhb84',
 'ips': [{'address': '123.123.123.123',
          'version': 4,
          'type: 'fixed',
          'meta': {...}},
         {'address': '124.124.124.124',
          'version': 4,
          'type': 'floating',
          'meta': {...}},
         {'address': 'fe80::4',
          'version': 6,
          'type': 'fixed',
          'meta': {...}}]
```

class VIF8021QbgParams(*managerid*, *typeid*, *typeidversion*, *instanceid*)
Bases: nova.network.model.Model

Represents the parameters for a 802.1qbg VIF.

class VIF8021QbhParams(*profileid*)
Bases: nova.network.model.Model

Represents the parameters for a 802.1qbh VIF.

ensure_string_keys(*d*)

get_netmask(*ip*, *subnet*)
Returns the netmask appropriate for injection into a guest.

3.7.477 The nova.network.neutronv2.api Module

class API(*skip_policy_check=False*)
Bases: nova.network.base_api.NetworkAPI

API for interacting with the neutron 2.x API.

add_dns_entry(*context*, *address*, *name*, *dns_type*, *domain*)
Create specified DNS entry for address.

add_fixed_ip_to_instance(*context*, **args*, ***kwargs*)
Add a fixed ip to the instance from specified network.

add_network_to_project(*context*, *project_id*, *network_uuid=None*)
Force add a network to the project.

allocate_floating_ip(*context*, *pool=None*)
Add a floating ip to a project from a pool.

allocate_for_instance(*context*, *instance*, ***kwargs*)
Allocate network resources for the instance.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **requested_networks** – optional value containing network_id, fixed_ip, and port_id
- **security_groups** – security groups to allocate for instance
- **macs** – None or a set of MAC addresses that the instance should use. macs is supplied by the hypervisor driver (contrast with requested_networks which is user supplied). NB: NeutronV2 currently assigns hypervisor supplied MAC addresses to arbitrary networks, which requires openflow switches to function correctly if more than one network is being used with the bare metal hypervisor (which is the only one known to limit MAC addresses).
- **dhcp_options** – None or a set of key/value pairs that should determine the DHCP BOOTP response, eg. for PXE booting an instance configured with the baremetal hypervisor. It is expected that these are already formatted for the neutron v2 api. See nova/virt/driver.py:dhcp_options_for_instance for an example.

allocate_port_for_instance(*context*, *instance*, *port_id*, *network_id=None*, *requested_ip=None*)
Allocate a port for the instance.

associate(*context*, *network_uuid*, *host=<object object at 0x7f858d3efe20>*, *project=<object object at 0x7f858d3efe20>*)
Associate a network for client.

associate_floating_ip(*context*, **args*, ***kwargs*)
Associate a floating ip with a fixed ip.

cleanup_instance_network_on_host(*context*, *instance*, *host*)
Cleanup network for specified instance on host.

create_pci_requests_for_sriov_ports(*context*, *pci_requests*, *requested_networks*)
Check requested networks for any SR-IOV port request.

Create a PCI request object for each SR-IOV port, and add it to the pci_requests object that contains a list of PCI request object.

create_private_dns_domain(*context*, *domain*, *availability_zone*)
Create a private DNS domain with nova availability zone.

create_public_dns_domain(*context*, *domain*, *project=None*)
Create a private DNS domain with optional nova project.

deallocate_for_instance(*context*, *instance*, ***kwargs*)
Deallocate all network resources related to the instance.

deallocate_port_for_instance(*context*, *instance*, *port_id*)
Remove a specified port from the instance.

Return network information for the instance

delete(*context*, *network_uuid*)
Delete a network for client.

delete_dns_domain(*context*, *domain*)
Delete the specified dns domain.

delete_dns_entry(*context*, *name*, *domain*)
Delete the specified dns entry.

disassociate(*context*, *network_uuid*)
Disassociate a network for client.

disassociate_and_release_floating_ip(*context*, *instance*, *floating_ip*)
Removes (deallocates) and deletes the floating ip.

This api call was added to allow this to be done in one operation if using neutron.

disassociate_floating_ip(*context*, **args*, ***kwargs*)
Disassociate a floating ip from the instance.

get(*context*, *network_uuid*)
Get specific network for client.

get_all(*context*)
Get all networks for client.

get_dns_domains(*context*)
Return a list of available dns domains.

These can be used to create DNS entries for floating ips.

get_dns_entries_by_address(*context*, *address*, *domain*)
Get entries for address and domain.

get_dns_entries_by_name(*context*, *name*, *domain*)
Get entries for name and domain.

get_fixed_ip(*context*, *id*)
Get a fixed ip from the id.

get_fixed_ip_by_address(*context*, *address*)
Return instance uuids given an address.

get_floating_ip(*context*, *id*)
Return floating ip object given the floating ip id.

get_floating_ip_by_address(*context*, *address*)
Return a floating ip given an address.

get_floating_ip_pools(*context*)
Return floating ip pool names.

get_floating_ips_by_project(*context*)

get_instance_id_by_floating_address(*context*, *address*)
Return the instance id a floating ip's fixed ip is allocated to.

get_vif_by_mac_address(*context*, *mac_address*)

get_vifs_by_instance(*context*, *instance*)

list_ports(*context*, ***search_opts*)
List ports for the client based on search options.

migrate_instance_finish(*context*, *instance*, *migration*)
Finish migrating the network of an instance.

migrate_instance_start(*context*, *instance*, *migration*)
Start to migrate the network of an instance.

modify_dns_entry(*context*, *name*, *address*, *domain*)
Create specified DNS entry for address.

release_floating_ip(*context*, *address*, *affect_auto_assigned=False*)
Remove a floating ip with the given address from a project.

remove_fixed_ip_from_instance(*context*, **args*, ***kwargs*)
Remove a fixed ip from the instance.

setup_instance_network_on_host(*context*, *instance*, *host*)
Setup network for specified instance on host.

setup_networks_on_host(*context*, *instance*, *host=None*, *teardown=False*)
Setup or teardown the network structures.

show_port(*context*, *port_id*)
Return the port for the client given the port id.

:param context - Request context. :param port_id - The id of port to be queried. :returns: A dict containing port data keyed by 'port', e.g.

```
{'port': {'port_id': 'abcd',
          'fixed_ip_address': '1.2.3.4'}}
```

validate_networks(*context*, *requested_networks*, *num_instances*)
Validate that the tenant can use the requested networks.

Return the number of instances than can be successfully allocated with the requested network configuration.

get_client(*context*, *admin=False*)

reset_state()

3.7.478 The nova.network.neutronv2.constants Module

3.7.479 The nova.network.noop_dns_driver Module

class NoopDNSDriver
Bases: nova.network.dns_driver.DNSDriver

No-op DNS manager. Does nothing.

create_domain(*_fqdomain*)

create_entry(*_name*, *_address*, *_type*, *_domain*)

delete_domain(*_fqdomain*)

delete_entry(*_name*, *_domain*)

get_domains()

get_entries_by_address(*_address*, *_domain*)

get_entries_by_name(*_name*, *_domain*)

modify_address(*_name*, *_address*, *_domain*)

3.7.480 The nova.network.opts Module

list_opts()

3.7.481 The nova.network.rpcapi Module

Client side of the network RPC API.

class NetworkAPI(*topic=None*)

Bases: object

Client side of the network rpc API.

API version history:

- 1.0 - Initial version.
- 1.1 - Adds migrate_instance_[start|finish]
- 1.2 - Make migrate_instance_[start|finish] a little more flexible
- 1.3 - Adds fanout cast update_dns for multi_host networks
- 1.4 - Add get_backdoor_port()
- 1.5 - Adds associate
- 1.6 - Adds instance_uuid to _{dis,}associate_floating_ip
- 1.7 - Adds method get_floating_ip_pools to replace get_floating_pools
- 1.8 - Adds macs to allocate_for_instance
- **1.9 - Adds rxtx_factor to [add|remove]_fixed_ip, removes** instance_uuid from allocate_for_instance and instance_get_nw_info

... Grizzly supports message version 1.9. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.9.

- 1.10- Adds (optional) requested_networks to deallocate_for_instance

... Havana supports message version 1.10. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.10.

- NOTE: remove unused method get_vifs_by_instance()
- NOTE: remove unused method get_vif_by_mac_address()
- NOTE: remove unused method get_network()
- NOTE: remove unused method get_all_networks()
- **1.11 - Add instance to deallocate_for_instance().** Remove instance_id, project_id, and host.
- 1.12 - Add instance to deallocate_fixed_ip()

... Icehouse supports message version 1.12. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.12.

- **1.13 - Convert allocate_for_instance()** to use NetworkRequestList objects

... Juno and Kilo supports message version 1.13. So, any changes to existing methods in 1.x after that point should be done such that they can handle the version_cap being set to 1.13.

- NOTE: remove unused method get_floating_ips_by_fixed_address()
- NOTE: remove unused method get_instance_uuids_by_ip_filter()
- NOTE: remove unused method disassociate_network()
- NOTE: remove unused method get_fixed_ip()
- NOTE: remove unused method get_fixed_ip_by_address()

- NOTE: remove unused method get_floating_ip()
- NOTE: remove unused method get_floating_ip_pools()
- NOTE: remove unused method get_floating_ip_by_address()
- NOTE: remove unused method get_floating_ips_by_project()
- NOTE: remove unused method get_instance_id_by_floating_address()
- NOTE: remove unused method allocate_floating_ip()
- NOTE: remove unused method deallocate_floating_ip()
- NOTE: remove unused method associate_floating_ip()
- NOTE: remove unused method disassociate_floating_ip()
- NOTE: remove unused method associate()
- 1.14 - Add mac parameter to release_fixed_ip().

VERSION_ALIASES = {'kilo': '1.13', 'grizzly': '1.9', 'havana': '1.10', 'juno': '1.13', 'icehouse': '1.12'}

add_dns_entry(*ctxt*, *address*, *name*, *dns_type*, *domain*)

add_fixed_ip_to_instance(*ctxt*, *instance_id*, *rxtx_factor*, *host*, *network_id*)

add_network_to_project(*ctxt*, *project_id*, *network_uuid*)

allocate_for_instance(*ctxt*, *instance_id*, *project_id*, *host*, *rxtx_factor*, *vpn*, *requested_networks*, *macs=None*, *dhcp_options=None*)

create_networks(*ctxt*, ***kwargs*)

create_private_dns_domain(*ctxt*, *domain*, *av_zone*)

create_public_dns_domain(*ctxt*, *domain*, *project*)

deallocate_fixed_ip(*ctxt*, *address*, *host*, *instance*)

deallocate_for_instance(*ctxt*, *instance*, *requested_networks=None*)

delete_dns_domain(*ctxt*, *domain*)

delete_dns_entry(*ctxt*, *name*, *domain*)

delete_network(*ctxt*, *uuid*, *fixed_range*)

get_dns_domains(*ctxt*)

get_dns_entries_by_address(*ctxt*, *address*, *domain*)

get_dns_entries_by_name(*ctxt*, *name*, *domain*)

get_instance_nw_info(*ctxt*, *instance_id*, *rxtx_factor*, *host*, *project_id*)

lease_fixed_ip(*ctxt*, *address*, *host*)

migrate_instance_finish(*ctxt*, *instance_uuid*, *rxtx_factor*, *project_id*, *source_compute*, *dest_compute*, *floating_addresses*, *host=None*)

migrate_instance_start(*ctxt*, *instance_uuid*, *rxtx_factor*, *project_id*, *source_compute*, *dest_compute*, *floating_addresses*, *host=None*)

modify_dns_entry(*ctxt*, *address*, *name*, *domain*)

release_fixed_ip(*ctxt*, *address*, *host*, *mac*)

remove_fixed_ip_from_instance(*ctxt*, *instance_id*, *rxtx_factor*, *host*, *address*)

rpc_setup_network_on_host (*ctxt*, *network_id*, *teardown*, *host*)

set_network_host (*ctxt*, *network_ref*)

setup_networks_on_host (*ctxt*, *instance_id*, *host*, *teardown*)

update_dns (*ctxt*, *network_ids*)

validate_networks (*ctxt*, *networks*)

3.7.482 The nova.network.security_group.neutron_driver Module

class SecurityGroupAPI (*skip_policy_check=False*)
Bases: nova.network.security_group.security_group_base.SecurityGroupBase

add_default_rules (*context*, *vals*)

add_rules (*context*, *id*, *name*, *vals*)
Add security group rule(s) to security group.

Note: the Nova security group API doesn't support adding multiple security group rules at once but the EC2 one does. Therefore, this function is written to support both. Multiple rules are installed to a security group in neutron using bulk support.

add_to_instance (*context*, *target*, **args*, ***kwargs*)
Add security group to the instance.

create_security_group (*context*, *name*, *description*)

default_rule_exists (*context*, *values*)

destroy (*context*, *security_group*)
This function deletes a security group.

get (*context*, *name=None*, *id=None*, *map_exception=False*)

get_all_default_rules (*context*)

get_default_rule (*context*, *id*)

get_instance_security_groups (*context*, *instance_uuid*, *detailed=False*)
Returns the security groups that are associated with an instance. If detailed is True then it also returns the full details of the security groups associated with an instance.

get_instances_security_groups_bindings (*context*, *servers*, *detailed=False*)
Returns a dict(instance_id, [security_groups]) to allow obtaining all of the instances and their security groups in one shot.

get_rule (*context*, *id*)

id_is_uuid = True

list (*context*, *names=None*, *ids=None*, *project=None*, *search_opts=None*)
Returns list of security group rules owned by tenant.

populate_security_groups (*instance*, *security_groups*)

remove_default_rules (*context*, *rule_ids*)

remove_from_instance (*context*, *target*, **args*, ***kwargs*)
Remove the security group associated with the instance.

remove_rules (*context*, *security_group*, *rule_ids*)

update_security_group (*context*, *security_group*, *name*, *description*)

validate_id(*id*)

3.7.483 The nova.network.security_group.openstack_driver Module

get_openstack_security_group_driver(*skip_policy_check=False*)

is_neutron_security_groups()

3.7.484 The nova.network.security_group.security_group_base Module

class SecurityGroupBase(*skip_policy_check=False*)

Bases: object

add_rules(*context*, *id*, *name*, *vals*)

add_to_instance(*context*, *instance*, *security_group_name*)

Add security group to the instance.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **security_group_name** – security group name to add

create_security_group(*context*, *name*, *description*)

create_security_group_rule(*context*, *security_group*, *new_rule*)

destroy(*context*, *security_group*)

ensure_default(*context*)

get(*context*, *name=None*, *id=None*, *map_exception=False*)

get_instance_security_groups(*context*, *instance_uuid*, *detailed=False*)

get_rule(*context*, *id*)

list(*context*, *names=None*, *ids=None*, *project=None*, *search_opts=None*)

static new_cidr_ingress_rule(*grantee_cidr*, *protocol*, *from_port*, *to_port*)

static new_group_ingress_rule(*grantee_group_id*, *protocol*, *from_port*, *to_port*)

parse_cidr(*cidr*)

populate_security_groups(*instance*, *security_groups*)

Called when populating the database for an instances security groups.

static raise_group_already_exists(*msg*)

static raise_invalid_cidr(*cidr*, *decoding_exception=None*)

static raise_invalid_group(*msg*)

static raise_invalid_property(*msg*)

static raise_not_found(*msg*)

static raise_over_quota(*msg*)

remove_from_instance(*context*, *instance*, *security_group_name*)

Remove the security group associated with the instance.

Parameters

- **context** – The request context.
- **instance** – nova.objects.instance.Instance object.
- **security_group_name** – security group name to remove

remove_rules(*context*, *security_group*, *rule_ids*)

rule_exists(*security_group*, *new_rule*)
Indicates whether the specified rule is already defined in the given security group.

trigger_handler(*event*, **args*)

trigger_members_refresh(*context*, *group_ids*)
Called when a security group gains a new or loses a member.

Sends an update request to each compute node for each instance for which this is relevant.

trigger_rules_refresh(*context*, *id*)
Called when a rule is added to or removed from a security_group.

update_security_group(*context*, *security_group*, *name*, *description*)

validate_property(*value*, *property*, *allowed*)

3.7.485 The nova.notifications Module

Functionality related to notifications common to multiple layers of the system.

audit_period_bounds(*current_period=False*)
Get the start and end of the relevant audit usage period

Parameters **current_period** – if True, this will generate a usage for the current usage period; if False, this will generate a usage for the previous audit period.

bandwidth_usage(*instance_ref*, *audit_start*, *ignore_missing_network_data=True*)
Get bandwidth usage information for the instance for the specified audit period.

image_meta(*system_metadata*)
Format image metadata for use in notifications from the instance system metadata.

info_from_instance(*context*, *instance*, *network_info*, *system_metadata*, ***kw*)
Get detailed instance information for an instance which is common to all notifications.

:param:instance: nova.objects.Instance :param:network_info: network_info provided if not None :param:system_metadata: system_metadata DB entries for the instance, if not None

---

Note: Currently unused here in trunk, but needed for potential custom modifications.

---

notify_decorator(*name*, *fn*)
Decorator for notify which is used from utils.monkey_patch().

Parameters

- **name** – name of the function
- **fn** –
  - object of the function

Returns fn – decorated function

send_api_fault(*url*, *status*, *exception*)
Send an api.fault notification.

send_update(*context*, *old_instance*, *new_instance*, *service='compute'*, *host=None*)
Send compute.instance.update notification to report any changes occurred in that instance

send_update_with_states(*context*, *instance*, *old_vm_state*, *new_vm_state*, *old_task_state*, *new_task_state*, *service='compute'*, *host=None*, *verify_states=False*)
Send compute.instance.update notification to report changes if there are any, in the instance

3.7.486 The nova.objects.agent Module

class Agent(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

architecture

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

destroy(**args*, ***kwargs*)

fields = {'deleted': Boolean(default=False,nullable=False), 'md5hash': String(default=<class 'oslo_versionedobjects.fie

classmethod get_by_triple(*context*, **args*, ***kwargs*)

hypervisor

id

md5hash

os

save(**args*, ***kwargs*)

updated_at

url

version

class AgentList(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.0'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

objects

3.7.487 The nova.objects.aggregate Module

class Aggregate(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

add_host(**args*, ***kwargs*)

availability_zone

create(**args*, ***kwargs*)

created_at

delete_host(**args*, ***kwargs*)

deleted

deleted_at

destroy(**args*, ***kwargs*)

fields = {'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'deleted': B

classmethod get_by_id(*context*, **args*, ***kwargs*)

hosts

id

metadata

name

obj_extra_fields = ['availability_zone']

save(**args*, ***kwargs*)

update_metadata(**args*, ***kwargs*)

updated_at

class AggregateList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.2'

child_versions = {'1.0': '1.1', '1.1': '1.1', '1.2': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

classmethod get_by_host(*context*, **args*, ***kwargs*)

classmethod get_by_metadata_key(*context*, **args*, ***kwargs*)

objects

3.7.488 The nova.objects.bandwidth_usage Module

class BandwidthUsage(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

bw_in

bw_out

create (*obj*, **args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'las

classmethod **get_by_instance_uuid_and_mac** (*obj*, **args*, ***kwargs*)

instance_uuid

last_ctr_in

last_ctr_out

last_refreshed

mac

start_period

updated_at

class BandwidthUsageList (**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.2'

child_versions = {'1.0': '1.0', '1.1': '1.1', '1.2': '1.2'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_by_uuids** (*obj*, **args*, ***kwargs*)

objects

3.7.489 The nova.objects.base Module

Nova common internal object model

class NovaObject (*context=None*, ***kwargs*)

Bases: object

Base class and object factory.

This forms the base of all objects that can be remoted or instantiated via RPC. Simply defining a class that inherits from this base class will make it remotely instantiatable. Objects should implement the necessary "get" classmethod routines as well as "save" object methods as appropriate.

VERSION = '1.0'

fields = {}

indirection_api = None

obj_alternate_context (**args*, ***kwds*)

obj_as_admin(**args*, ***kwds*)
Context manager to make an object call as an admin.

This temporarily modifies the context embedded in an object to be elevated() and restores it after the call completes. Example usage:

with obj.obj_as_admin(): obj.save()

obj_attr_is_set(*attrname*)
Test object to see if attrname is present.

Returns True if the named attribute has a value set, or False if not. Raises AttributeError if attrname is not a valid attribute for this object.

obj_calculate_child_version(*target_version*, *child*)
Calculate the appropriate version for a child object.

This is to be used when backporting an object for an older client. A sub-object will need to be backported to a suitable version for the client as well, and this method will calculate what that version should be, based on obj_relationships.

Parameters

- **target_version** – Version this object is being backported to
- **child** – The child field for which the appropriate version is to be calculated

Returns None if the child should be omitted from the backport, otherwise, the version to which the child should be backported

classmethod obj_class_from_name(*objname*, *objver*)
Returns a class from the registry based on a name and version.

obj_clone()
Create a copy.

obj_extra_fields = []

obj_fields

classmethod obj_from_primitive(*primitive*, *context=None*)
Object field-by-field hydration.

obj_get_changes()
Returns a dict of changed fields and their new values.

obj_load_attr(*attrname*)
Load an additional attribute from the real object.

This should use self._conductor, and cache any data that might be useful for future load operations.

obj_make_compatible(*primitive*, *target_version*)
Make an object representation compatible with a target version.

This is responsible for taking the primitive representation of an object and making it suitable for the given target_version. This may mean converting the format of object attributes, removing attributes that have been added since the target version, etc. In general:

•If a new version of an object adds a field, this routine should remove it for older versions.

•If a new version changed or restricted the format of a field, this should convert it back to something a client knowing only of the older version will tolerate.

•If an object that this object depends on is bumped, then this object should also take a version bump. Then, this routine should backlevel the dependent object (by calling its obj_make_compatible()) if the requested version of this object is older than the version where the new dependent object was added.

:param:primitive: The result of self.obj_to_primitive() :param:target_version: The version string requested by the recipient of the object :raises: nova.exception.UnsupportedObjectError if conversion is not possible for some reason

classmethod obj_name()
Return a canonical name for this object which will be used over the wire for remote hydration.

obj_relationships = {}

obj_reset_changes(*fields=None*, *recursive=False*)
Reset the list of fields that have been changed.

Parameters

- **fields** – List of fields to reset, or "all" if None.
- **recursive** – Call obj_reset_changes(recursive=True) on any sub-objects within the list of fields being reset.

NOTE: This is NOT "revert to previous values" NOTE: Specifying fields on recursive resets will only be honored at the top level. Everything below the top will reset all.

obj_set_defaults(**attrs*)

obj_to_primitive(*target_version=None*)
Simple base-case dehydration.

This calls to_primitive() for each item in fields.

obj_what_changed()
Returns a set of fields that have been modified.

save(*context*)
Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

class NovaObjectDictCompat
Bases: `oslo_versionedobjects.base.VersionedObjectDictCompat`

keys()

class NovaObjectRegistry
Bases: `oslo_versionedobjects.base.VersionedObjectRegistry`

registration_hook(*cls*, *index*)

class NovaObjectSerializer
Bases: `oslo_messaging.serializer.NoOpSerializer`

A NovaObject-aware Serializer.

This implements the Oslo Serializer interface and provides the ability to serialize and deserialize NovaObject entities. Any service that needs to accept or return NovaObjects as arguments or result values should pass this to its RPCClient and RPCServer objects.

conductor

deserialize_entity(*context*, *entity*)

serialize_entity(*context*, *entity*)

class NovaPersistentObject
Bases: object

Mixin class for Persistent objects.

This adds the fields that we use in common for most persistent objects.

fields = {'deleted': Boolean(default=False,nullable=False), 'created_at': DateTime(default=<class 'oslo_versionedobjec

class NovaTimestampObject
Bases: object

Mixin class for db backed objects with timestamp fields.

Sqlalchemy models that inherit from the oslo_db TimestampMixin will include these fields and the corresponding objects will benefit from this mixin.

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'up

class ObjectListBase(**args*, ***kwargs*)
Bases: oslo_versionedobjects.base.ObjectListBase

get_attrname(*name*)
Return the mangled name of the attribute's underlying storage.

obj_equal_prims(*obj_1*, *obj_2*, *ignore=None*)
Compare two primitives for equivalence ignoring some keys.

This operation tests the primitives of two objects for equivalence. Object primitives may contain a list identifying fields that have been changed - this is ignored in the comparison. The ignore parameter lists any other keys to be ignored.

:param:obj1: The first object in the comparison :param:obj2: The second object in the comparison :param:ignore: A list of fields to ignore :returns: True if the primitives are equal ignoring changes and specified fields, otherwise False.

obj_make_list(*context*, *list_obj*, *item_cls*, *db_list*, ***extra_args*)
Construct an object list from a list of primitives.

This calls item_cls._from_db_object() on each item of db_list, and adds the resulting object to list_obj.

:param:context: Request context :param:list_obj: An ObjectListBase object :param:item_cls: The NovaObject class of the objects within the list :param:db_list: The list of primitives to convert to objects :param:extra_args: Extra arguments to pass to _from_db_object() :returns: list_obj

obj_to_primitive(*obj*)
Recursively turn an object into a python primitive.

A NovaObject becomes a dict, and anything that implements ObjectListBase becomes a list.

remotable(*fn*)
Decorator for remotable object methods.

remotable_classmethod(*fn*)
Decorator for remotable classmethods.

serialize_args(*fn*)
Decorator that will do the arguments serialization before remoting.

3.7.490 The nova.objects.block_device Module

class BlockDeviceMapping(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject,

nova.objects.base.NovaObjectDictCompat

VERSION = '1.13'

boot_index

connection_info

create(**args*, ***kwargs*)

created_at

delete_on_termination

deleted

deleted_at

destination_type

destroy(**args*, ***kwargs*)

device_name

device_type

disk_bus

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'gu

classmethod **get_by_volume_id**(*context*, **args*, ***kwargs*)

get_image_mapping()

guest_format

id

image_id

instance

instance_uuid

is_image

is_root

is_volume

no_device

obj_load_attr(*attrname*)

obj_relationships = {'instance': [('1.0', '1.13'), ('1.2', '1.14'), ('1.3', '1.15'), ('1.4', '1.16'), ('1.5', '1.17'), ('1.6', '1.18

save(**args*, ***kwargs*)

snapshot_id

source_type

update_or_create(**args*, ***kwargs*)

updated_at

volume_id

volume_size

class BlockDeviceMappingList (**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.14'

child_versions = {'1.4': '1.3', '1.5': '1.4', '1.6': '1.5', '1.7': '1.6', '1.0': '1.0', '1.1': '1.1', '1.2': '1.1', '1.3': '1.2', '1.8'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_by_instance_uuid (*context*, **args*, ***kwargs*)

objects

root_bdm ()

root_metadata (*context*, *image_api*, *volume_api*)

block_device_make_list (*context*, *db_list*, ***extra_args*)

block_device_make_list_from_dicts (*context*, *bdm_dicts_list*)

3.7.491 The nova.objects.cell_mapping Module

class CellMapping (*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaTimestampObject, nova.objects.base.NovaObject

VERSION = '1.0'

create (**args*, ***kwargs*)

created_at

database_connection

destroy (**args*, ***kwargs*)

fields = {'uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'transport_u

classmethod get_by_uuid (*context*, **args*, ***kwargs*)

id

name

save (**args*, ***kwargs*)

transport_url

updated_at

uuid

3.7.492 The nova.objects.compute_node Module

class ComputeNode (*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.11'

cpu_info

create (**args*, ***kwargs*)

created_at

current_workload

deleted

deleted_at

destroy(**args*, ***kwargs*)

disk_available_least

fields = {'pci_device_pools': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),

free_disk_gb

free_ram_mb

classmethod get_by_host_and_nodename(*context*, **args*, ***kwargs*)

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_service_id(*context*, **args*, ***kwargs*)

classmethod get_first_node_by_host_for_old_compat(*context*, **args*, ***kwargs*)

host

host_ip

hypervisor_hostname

hypervisor_type

hypervisor_version

id

local_gb

local_gb_used

memory_mb

memory_mb_used

metrics

numa_topology

obj_make_compatible(*primitive*, *target_version*)

obj_relationships = {'supported_hv_specs': [('1.6', '1.0')], 'pci_device_pools': [('1.9', '1.0'), ('1.11', '1.1')]}

pci_device_pools

running_vms

save(**args*, ***kwargs*)

service

service_id

stats

supported_hv_specs

update_from_virt_driver(*resources*)

updated_at

vcpus

vcpus_used

class ComputeNodeList (**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.11'

child_versions = {'1.4': '1.5', '1.5': '1.5', '1.6': '1.6', '1.7': '1.7', '1.0': '1.2', '1.1': '1.3', '1.2': '1.3', '1.3': '1.4', '1.8'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all (*context*, **args*, ***kwargs*)

classmethod get_all_by_host (*context*, **args*, ***kwargs*)

classmethod get_by_hypervisor (*context*, **args*, ***kwargs*)

classmethod get_by_service (*context*, *service*, *use_slave=False*)

objects

3.7.493 The nova.objects.dns_domain Module

class DNSDomain (*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

availability_zone

created_at

classmethod delete_by_domain (*context*, **args*, ***kwargs*)

deleted

deleted_at

domain

fields = {'domain': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'deleted_a

classmethod get_by_domain (*context*, **args*, ***kwargs*)

project_id

classmethod register_for_project (*context*, **args*, ***kwargs*)

classmethod register_for_zone (*context*, **args*, ***kwargs*)

scope

updated_at

class DNSDomainList (**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.0'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all (*context*, **args*, ***kwargs*)

objects

3.7.494 The nova.objects.ec2 Module

class EC2Ids(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject

VERSION = '1.0'

ami_id

fields = {'instance_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'kerne

classmethod get_by_instance(*context*, **args*, ***kwargs*)

instance_id

kernel_id

ramdisk_id

class EC2InstanceMapping(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'deleted': Bo

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

id

updated_at

uuid

class EC2SnapshotMapping(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'deleted': Bo

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

id

updated_at

uuid

class EC2VolumeMapping(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'deleted': Bo

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

id

updated_at

uuid

class S3ImageMapping(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'deleted': Bo

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

id

updated_at

uuid

3.7.495 The nova.objects.external_event Module

class InstanceExternalEvent(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

data

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'sta

instance_uuid

key

static **make_key**(*name*, *tag=None*)

name

status

tag

3.7.496 The nova.objects.fields Module

class Architecture(***kwargs*)
Bases: oslo_versionedobjects.fields.Enum

coerce(*obj*, *attr*, *value*)

class ArchitectureField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.Architecture object at 0x7f858bef7490>

class AutoTypedField(***kwargs*)
Bases: oslo_versionedobjects.fields.Field

AUTO_TYPE = None

class BaseEnumField(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

This class should not be directly instantiated. Instead subclass it and set AUTO_TYPE to be a SomeEnum() where SomeEnum is a subclass of Enum.

class BlockDeviceDestinationType
Bases: oslo_versionedobjects.fields.Enum

Represents possible destination_type values for a BlockDeviceMapping.

ALL = ('local', 'volume')

LOCAL = 'local'

VOLUME = 'volume'

class BlockDeviceDestinationTypeField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.BlockDeviceDestinationType object at 0x7f858bef7550>

class BlockDeviceSourceType
Bases: oslo_versionedobjects.fields.Enum

Represents the possible source_type values for a BlockDeviceMapping.

ALL = ('blank', 'image', 'snapshot', 'volume')

BLANK = 'blank'

IMAGE = 'image'

SNAPSHOT = 'snapshot'

VOLUME = 'volume'

class BlockDeviceSourceTypeField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.BlockDeviceSourceType object at 0x7f858bef75d0>

class BlockDeviceType
Bases: oslo_versionedobjects.fields.Enum

Represents possible device_type values for a BlockDeviceMapping.

ALL = ('cdrom', 'disk', 'floppy', 'fs', 'lun')

CDROM = 'cdrom'

DISK = 'disk'

FLOPPY = 'floppy'

FS = 'fs'

LUN = 'lun'

class BlockDeviceTypeField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.BlockDeviceType object at 0x7f858bef7610>

class CPUAllocationPolicy
Bases: oslo_versionedobjects.fields.Enum

ALL = ('dedicated', 'shared')

DEDICATED = 'dedicated'

SHARED = 'shared'

class CPUAllocationPolicyField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.CPUAllocationPolicy object at 0x7f858bef7690>

class CPUFeaturePolicy(***kwargs*)
Bases: oslo_versionedobjects.fields.Enum

class CPUFeaturePolicyField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.CPUFeaturePolicy object at 0x7f858bef7850>

class CPUMatch(***kwargs*)
Bases: oslo_versionedobjects.fields.Enum

class CPUMatchField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.CPUMatch object at 0x7f858bef77d0>

class CPUMode(***kwargs*)
Bases: oslo_versionedobjects.fields.Enum

class CPUModeField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.CPUMode object at 0x7f858bef7710>

class DiskBus
Bases: oslo_versionedobjects.fields.Enum

ALL = ('fdc', 'ide', 'sata', 'scsi', 'usb', 'virtio', 'xen')

FDC = 'fdc'

IDE = 'ide'

SATA = 'sata'

SCSI = 'scsi'

USB = 'usb'

VIRTIO = 'virtio'

XEN = 'xen'

class **DiskBusField**(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.DiskBus object at 0x7f858bef78d0>

class **FlexibleBoolean**
Bases: oslo_versionedobjects.fields.Boolean

static **coerce**(*obj*, *attr*, *value*)

class **FlexibleBooleanField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.FlexibleBoolean object at 0x7f858bef7d90>

class **HVType**
Bases: oslo_versionedobjects.fields.Enum

coerce(*obj*, *attr*, *value*)

class **HVTypeField**(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.HVType object at 0x7f858bef7950>

class **IPAddress**
Bases: oslo_versionedobjects.fields.FieldType

static **coerce**(*obj*, *attr*, *value*)

from_primitive(*obj*, *attr*, *value*)

static **to_primitive**(*obj*, *attr*, *value*)

class **IPAddressField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPAddress object at 0x7f858bef7e10>

class **IPNetwork**
Bases: nova.objects.fields.IPAddress

static **coerce**(*obj*, *attr*, *value*)

class **IPNetworkField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPNetwork object at 0x7f858be83050>

class **IPV4Address**
Bases: nova.objects.fields.IPAddress

static **coerce**(*obj*, *attr*, *value*)

class **IPV4AddressField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPV4Address object at 0x7f858bef7e90>

class **IPV4AndV6Address**
Bases: nova.objects.fields.IPAddress

static coerce(*obj*, *attr*, *value*)

class **IPV4AndV6AddressField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPV4AndV6Address object at 0x7f858bef7f90>

class **IPV4Network**
Bases: nova.objects.fields.IPNetwork

static coerce(*obj*, *attr*, *value*)

class **IPV4NetworkField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPV4Network object at 0x7f858be830d0>

class **IPV6Address**
Bases: nova.objects.fields.IPAddress

static coerce(*obj*, *attr*, *value*)

class **IPV6AddressField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPV6Address object at 0x7f858bef7f10>

class **IPV6Network**
Bases: nova.objects.fields.IPNetwork

static coerce(*obj*, *attr*, *value*)

class **IPV6NetworkField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <nova.objects.fields.IPV6Network object at 0x7f858be83150>

class **ListOfIntegersField**(***kwargs*)
Bases: nova.objects.fields.AutoTypedField

AUTO_TYPE = <oslo_versionedobjects.fields.List object at 0x7f858be83250>

class **ListOfObjectsField**(*objtype*, ***kwargs*)
Bases: nova.objects.fields.AutoTypedField

class **MonitorMetricType**
Bases: oslo_versionedobjects.fields.Enum

ALL = ('cpu.frequency', 'cpu.user.time', 'cpu.kernel.time', 'cpu.idle.time', 'cpu.iowait.time', 'cpu.user.percent', 'cpu.kern

CPU_FREQUENCY = 'cpu.frequency'

CPU_IDLE_PERCENT = 'cpu.idle.percent'

CPU_IDLE_TIME = 'cpu.idle.time'

CPU_IOWAIT_PERCENT = 'cpu.iowait.percent'

CPU_IOWAIT_TIME = 'cpu.iowait.time'

CPU_KERNEL_PERCENT = 'cpu.kernel.percent'

CPU_KERNEL_TIME = 'cpu.kernel.time'

CPU_PERCENT = 'cpu.percent'

CPU_USER_PERCENT = 'cpu.user.percent'

CPU_USER_TIME = 'cpu.user.time'

class MonitorMetricTypeField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.MonitorMetricType object at 0x7f858bef7d10>

class NetworkModel
Bases: oslo_versionedobjects.fields.FieldType

static coerce(*obj*, *attr*, *value*)

static from_primitive(*obj*, *attr*, *value*)

stringify(*value*)

static to_primitive(*obj*, *attr*, *value*)

class OSType
Bases: oslo_versionedobjects.fields.Enum

ALL = ('linux', 'windows')

LINUX = 'linux'

WINDOWS = 'windows'

coerce(*obj*, *attr*, *value*)

class OSTypeField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.OSType object at 0x7f858bef7990>

class Object(*obj_name*, ***kwargs*)
Bases: oslo_versionedobjects.fields.FieldType

coerce(*obj*, *attr*, *value*)

describe()

static from_primitive(*obj*, *attr*, *value*)

stringify(*value*)

static to_primitive(*obj*, *attr*, *value*)

class ObjectField(*objtype*, ***kwargs*)
Bases: nova.objects.fields.AutoTypedField

class RNGModel
Bases: oslo_versionedobjects.fields.Enum

ALL = ('virtio',)

VIRTIO = 'virtio'

class RNGModelField(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.RNGModel object at 0x7f858bef7a10>

class SCSIModel
Bases: oslo_versionedobjects.fields.Enum

ALL = ('buslogic', 'ibmvscsi', 'lsilogic', 'lsisas1068', 'lsisas1078', 'virtio-scsi', 'vmpvscsi')

BUSLOGIC = 'buslogic'

IBMVSCSI = 'ibmvscsi'

LSILOGIC = 'lsilogic'

LSISAS1068 = 'lsisas1068'

LSISAS1078 = 'lsisas1078'

VIRTIO_SCSI = 'virtio-scsi'

VMPVSCSI = 'vmpvscsi'

coerce(*obj*, *attr*, *value*)

class **SCSIModelField**(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.SCSIModel object at 0x7f858bef7a90>

class **VIFModel**
Bases: oslo_versionedobjects.fields.Enum

LEGACY_VALUES = {'virtuale1000e': 'e1000e', 'virtualvmxnet3': 'vmxnet3', 'virtualvmxnet': 'vmxnet', 'virtualpcnet32'

coerce(*obj*, *attr*, *value*)

class **VIFModelField**(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.VIFModel object at 0x7f858bef7b90>

class **VMMode**
Bases: oslo_versionedobjects.fields.Enum

coerce(*obj*, *attr*, *value*)

class **VMModeField**(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.VMMode object at 0x7f858bef7c10>

class **VideoModel**
Bases: oslo_versionedobjects.fields.Enum

ALL = ('cirrus', 'qxl', 'vga', 'vmvga', 'xen')

CIRRUS = 'cirrus'

QXL = 'qxl'

VGA = 'vga'

VMVGA = 'vmvga'

XEN = 'xen'

class **VideoModelField**(***kwargs*)
Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.VideoModel object at 0x7f858bef7b10>

class **WatchdogAction**
Bases: oslo_versionedobjects.fields.Enum

ALL = ('none', 'pause', 'poweroff', 'reset')

NONE = 'none'

PAUSE = 'pause'

POWEROFF = 'poweroff'

RESET = 'reset'

class **WatchdogActionField**(***kwargs*)

Bases: nova.objects.fields.BaseEnumField

AUTO_TYPE = <nova.objects.fields.WatchdogAction object at 0x7f858bef7c90>

3.7.497 The nova.objects.fixed_ip Module

class **FixedIP**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.11'

address

allocated

classmethod **associate**(*context*, **args*, ***kwargs*)

classmethod **associate_pool**(*context*, **args*, ***kwargs*)

create(**args*, ***kwargs*)

created_at

default_route

deleted

deleted_at

disassociate(**args*, ***kwargs*)

classmethod **disassociate_all_by_timeout**(*context*, *host*, *time*)

classmethod **disassociate_by_address**(*context*, **args*, ***kwargs*)

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'res

floating_ips

classmethod **get_by_address**(*context*, **args*, ***kwargs*)

classmethod **get_by_floating_address**(*context*, **args*, ***kwargs*)

classmethod **get_by_id**(*context*, **args*, ***kwargs*)

classmethod **get_by_network_and_host**(*context*, **args*, ***kwargs*)

host

id

instance

instance_uuid

leased

network

network_id

obj_make_compatible(*primitive*, *target_version*)

obj_relationships = {'instance': [('1.0', '1.13'), ('1.2', '1.14'), ('1.3', '1.15'), ('1.6', '1.16'), ('1.7', '1.17'), ('1.8', '1.18

reserved

save(**args*, ***kwargs*)

updated_at

virtual_interface

virtual_interface_id

class **FixedIPList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.11'

classmethod **bulk_create**(*context*, **args*, ***kwargs*)

child_versions = {'1.4': '1.4', '1.5': '1.5', '1.6': '1.6', '1.7': '1.7', '1.0': '1.0', '1.1': '1.1', '1.2': '1.2', '1.3': '1.3', '1.8':

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_all**(*context*, **args*, ***kwargs*)

classmethod **get_by_host**(*context*, **args*, ***kwargs*)

classmethod **get_by_instance_uuid**(*context*, **args*, ***kwargs*)

classmethod **get_by_network**(*context*, **args*, ***kwargs*)

classmethod **get_by_virtual_interface_id**(*context*, **args*, ***kwargs*)

objects

3.7.498 The nova.objects.flavor Module

class **Flavor**(**args*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

add_access(**args*, ***kwargs*)

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

destroy(**args*, ***kwargs*)

disabled

ephemeral_gb

extra_specs

fields = {'memory_mb': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'roo

flavorid

classmethod **get_by_flavor_id**(*context*, **args*, ***kwargs*)

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_name(*context*, **args*, ***kwargs*)

id

is_public

memory_mb

name

obj_load_attr(*attrname*)

obj_reset_changes(*fields=None*)

obj_what_changed()

projects

remove_access(**args*, ***kwargs*)

root_gb

rxtx_factor

save()

save_extra_specs(**args*, ***kwargs*)
Add or delete extra_specs.

:param:to_add: A dict of new keys to add/update :param:to_delete: A list of keys to remove

save_projects(**args*, ***kwargs*)
Add or delete projects.

:param:to_add: A list of projects to add :param:to_delete: A list of projects to remove

swap

updated_at

vcpu_weight

vcpus

class FlavorList(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

child_versions = {'1.0': '1.0', '1.1': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

objects

3.7.499 The nova.objects.floating_ip Module

class FloatingIP(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.7'

address

classmethod allocate_address(*context*, **args*, ***kwargs*)

classmethod associate(*context*, **args*, ***kwargs*)

auto_assigned

created_at

classmethod deallocate(*context*, **args*, ***kwargs*)

deleted

deleted_at

classmethod destroy(*context*, **args*, ***kwargs*)

classmethod disassociate(*context*, **args*, ***kwargs*)

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'del

fixed_ip

fixed_ip_id

classmethod get_addresses_by_instance(*context*, *instance*)

classmethod get_by_address(*context*, **args*, ***kwargs*)

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_pool_names(*context*, **args*, ***kwargs*)

host

id

interface

obj_load_attr(*attrname*)

obj_relationships = {'fixed_ip': [('1.0', '1.1'), ('1.2', '1.2'), ('1.3', '1.3'), ('1.4', '1.4'), ('1.5', '1.5'), ('1.6', '1.6'), ('1.7

pool

project_id

save(**args*, ***kwargs*)

updated_at

class FloatingIPList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.8'

child_versions = {'1.4': '1.3', '1.5': '1.4', '1.6': '1.5', '1.7': '1.6', '1.0': '1.0', '1.1': '1.1', '1.2': '1.1', '1.3': '1.2', '1.8'

classmethod create(*context*, **args*, ***kwargs*)

classmethod destroy(*context*, **args*, ***kwargs*)

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

classmethod get_by_fixed_address(*context*, **args*, ***kwargs*)

classmethod get_by_fixed_ip_id(*context*, **args*, ***kwargs*)

classmethod get_by_host(*context*, **args*, ***kwargs*)

classmethod get_by_project (*context*, **args*, ***kwargs*)

static make_ip_info (*address*, *pool*, *interface*)

objects

3.7.500 The nova.objects.host_mapping Module

class HostMapping (*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaTimestampObject, nova.objects.base.NovaObject

VERSION = '1.0'

cell_mapping

create (**args*, ***kwargs*)

created_at

destroy (**args*, ***kwargs*)

fields = {'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cr

classmethod get_by_host (*context*, **args*, ***kwargs*)

host

id

obj_load_attr (*attrname*)

obj_relationships = {'cell_mapping': [('1.0', '1.0')]}

save (**args*, ***kwargs*)

updated_at

3.7.501 The nova.objects.hv_spec Module

class HVSpec (*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

arch

fields = {'vm_mode': VMMode(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False,valid_

classmethod from_list (*data*)

hv_type

to_list ()

vm_mode

3.7.502 The nova.objects.image_meta Module

class ImageMeta (*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject

VERSION = '1.1'

checksum

container_format

created_at

direct_url

disk_format

fields = {'status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'tags': List(

classmethod from_dict(*image_meta*)
Create instance from image metadata dict

Parameters **image_meta** – image metadata dictionary

Creates a new object instance, initializing from the properties associated with the image metadata instance

Returns an ImageMeta instance

classmethod from_instance(*instance*)
Create instance from instance system metadata

Parameters **instance** – Instance object

Creates a new object instance, initializing from the system metadata "image_*" properties associated with instance

Returns an ImageMeta instance

id

min_disk

min_ram

name

obj_relationships = {'properties': [('1.0', '1.0'), ('1.1', '1.1')]}

owner

properties

protected

size

status

tags

updated_at

virtual_size

visibility

class ImageMetaProps(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject

NUMA_NODES_MAX = 128

VERSION = '1.1'

fields = {'img_cache_in_nova': FlexibleBoolean(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nulla

classmethod from_dict(*image_props*)
Create instance from image properties dict

Parameters **image_props** – dictionary of image metdata properties

Creates a new object instance, initializing from a dictionary of image metadata properties

Returns an ImageMetaProps instance

get(*name*, *defvalue=None*)

Get the value of an attribute :param name: the attribute to request :param defvalue: the default value if not set

This returns the value of an attribute if it is currently set, otherwise it will return None.

This differs from accessing props.attrname, because that will raise an exception if the attribute has no value set.

So instead of

if image_meta.properties.obj_attr_is_set("some_attr"): val = image_meta.properties.some_attr

else val = None

Callers can rely on unconditional access

val = image_meta.properties.get("some_attr")

Returns the attribute value or None

hw_architecture

hw_auto_disk_config

hw_boot_menu

hw_cdrom_bus

hw_cpu_cores

hw_cpu_max_cores

hw_cpu_max_sockets

hw_cpu_max_threads

hw_cpu_policy

hw_cpu_sockets

hw_cpu_threads

hw_device_id

hw_disk_bus

hw_disk_type

hw_floppy_bus

hw_ipxe_boot

hw_machine_type

hw_mem_page_size

hw_numa_cpus

hw_numa_mem

hw_numa_nodes

hw_qemu_guest_agent

hw_rng_model

hw_scsi_model

hw_serial_port_count

hw_video_model

hw_video_ram

hw_vif_model

hw_vm_mode

hw_watchdog_action

img_bdm_v2

img_bittorrent

img_block_device_mapping

img_cache_in_nova

img_compression_level

img_linked_clone

img_mappings

img_owner_id

img_root_device_name

img_use_agent

img_version

os_command_line

os_distro

os_require_quiesce

os_skip_agent_inject_files_at_boot

os_skip_agent_inject_ssh

os_type

3.7.503 The nova.objects.instance Module

class Instance (**args*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.21'

access_ip_v4

access_ip_v6

architecture

auto_disk_config

availability_zone

cell_name

cleaned

config_drive

create(**args*, ***kwargs*)

created_at

default_ephemeral_device

default_swap_device

delete_flavor(*namespace*)

delete_metadata_key(**args*, ***kwargs*)
Optimized metadata delete method.

This provides a more efficient way to delete a single metadata key, instead of just calling instance.save(). This should be called with the key still present in self.metadata, which it will update after completion.

deleted

deleted_at

destroy(**args*, ***kwargs*)

disable_terminate

display_description

display_name

ec2_ids

ephemeral_gb

ephemeral_key_uuid

fault

fields = {'vm_state': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'pci_req

flavor

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

get_flavor(*namespace=None*)

host

hostname

id

image_ref

info_cache

instance_type_id

kernel_id

key_data

key_name

launch_index

launched_at

launched_on

locked

locked_by

memory_mb

metadata

name

new_flavor

node

numa_topology

obj_extra_fields = ['name']

obj_load_attr(*attrname*)

obj_make_compatible(*primitive*, *target_version*)

obj_relationships = {'pci_devices': [('1.6', '1.0'), ('1.15', '1.1')], 'pci_requests': [('1.16', '1.1')], 'tags': [('1.17', '1.0'

obj_reset_changes(*fields=None*)

obj_what_changed()

old_flavor

os_type

pci_devices

pci_requests

power_state

progress

project_id

ramdisk_id

refresh(**args*, ***kwargs*)

reservation_id

root_device_name

root_gb

save(**args*, ***kwargs*)

Save updates to this instance

Column-wise updates will be made based on the result of self.what_changed(). If expected_task_state is provided, it will be checked against the in-database copy of the instance before updates are made.

:param:context: Security context :param:expected_task_state: Optional tuple of valid task states for the instance to be in :param:expected_vm_state: Optional tuple of valid vm states for the instance to be in :param admin_state_reset: True if admin API is forcing setting of task_state/vm_state

scheduled_at

security_groups

set_flavor(*flavor*, *namespace=None*)

shutdown_terminate

skip_cells_sync(**args*, ***kwds*)
Context manager to save an instance without syncing cells.

Temporarily disables the cells syncing logic, if enabled. This should only be used when saving an instance that has been passed down/up from another cell in order to avoid passing it back to the originator to be re-saved.

system_metadata

tags

task_state

terminated_at

updated_at

user_data

user_id

uuid

vcpu_model

vcpus

vm_mode

vm_state

class InstanceList(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.19'

child_versions = {'1.4': '1.12', '1.5': '1.12', '1.6': '1.13', '1.7': '1.13', '1.12': '1.16', '1.1': '1.9', '1.2': '1.11', '1.3': '1.

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

fill_faults()
Batch query the database for our instances' faults.

Returns A list of instance uuids for which faults were found.

classmethod get_active_by_window_joined(*context*, *begin*, *end=None*, *project_id=None*, *host=None*, *expected_attrs=None*, *use_slave=False*)
Get instances and joins active during a certain time window.

:param:context: nova request context :param:begin: datetime for the start of the time window :param:end: datetime for the end of the time window :param:project_id: used to filter instances by project :param:host: used to filter instances on a given compute host :param:expected_attrs: list of related fields that can be joined in the database layer when querying for instances :param use_slave if True, ship this query off to a DB slave :returns: InstanceList

classmethod get_all(*context*, **args*, ***kwargs*)
Returns all instances on all nodes.

classmethod get_by_filters(*context*, **args*, ***kwargs*)

classmethod get_by_host(*context*, **args*, ***kwargs*)

classmethod get_by_host_and_node(*context*, **args*, ***kwargs*)

classmethod get_by_host_and_not_type(*context*, **args*, ***kwargs*)

classmethod get_by_security_group(*context*, *security_group*)

classmethod get_by_security_group_id(*context*, **args*, ***kwargs*)

objects

3.7.504 The nova.objects.instance_action Module

class InstanceAction(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

action

classmethod action_finish(*context*, **args*, ***kwargs*)

classmethod action_start(*context*, **args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cre

finish(**args*, ***kwargs*)

finish_time

classmethod get_by_request_id(*context*, **args*, ***kwargs*)

id

instance_uuid

message

static pack_action_finish(*context*, *instance_uuid*)

static pack_action_start(*context*, *instance_uuid*, *action_name*)

project_id

request_id

start_time

updated_at

user_id

class InstanceActionEvent(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

action_id

created_at

deleted

deleted_at

event

classmethod event_finish(*context*, **args*, ***kwargs*)

classmethod event_finish_with_failure(*obj*, **args*, ***kwargs*)

classmethod event_start(*context*, **args*, ***kwargs*)

fields = {'deleted': Boolean(default=False,nullable=False), 'start_time': DateTime(default=<class 'oslo_versionedobjec

finish(**args*, ***kwargs*)

finish_time

finish_with_failure(**args*, ***kwargs*)

classmethod get_by_id(*context*, **args*, ***kwargs*)

id

static pack_action_event_finish(*context*, *instance_uuid*, *event_name*, *exc_val=None*, *exc_tb=None*)

static pack_action_event_start(*context*, *instance_uuid*, *event_name*)

result

start_time

traceback

updated_at

class InstanceActionEventList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

child_versions = {'1.0': '1.0', '1.1': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_by_action(*context*, **args*, ***kwargs*)

objects

class InstanceActionList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_by_instance_uuid(*context*, **args*, ***kwargs*)

objects

3.7.505 The nova.objects.instance_fault Module

class InstanceFault(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

code

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

details

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'cre

classmethod **get_latest_for_instance**(*context*, **args*, ***kwargs*)

host

id

instance_uuid

message

updated_at

class **InstanceFaultList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

child_versions = {'1.0': '1.1', '1.1': '1.2'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_by_instance_uuids**(*context*, **args*, ***kwargs*)

objects

3.7.506 The nova.objects.instance_group Module

class **InstanceGroup**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.9'

classmethod **add_members**(*context*, **args*, ***kwargs*)

count_members_by_user(**args*, ***kwargs*)

Count the number of instances in a group belonging to a user.

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

destroy(**args*, ***kwargs*)

fields = {'deleted': Boolean(default=False,nullable=False), 'updated_at': DateTime(default=<class 'oslo_versionedobje

classmethod **get_by_hint**(*context*, *hint*)

classmethod **get_by_instance_uuid**(*context*, **args*, ***kwargs*)

classmethod **get_by_name**(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

get_hosts(**args*, ***kwargs*)
Get a list of hosts for non-deleted instances in the group

This method allows you to get a list of the hosts where instances in this group are currently running. There's also an option to exclude certain instance UUIDs from this calculation.

id

members

name

obj_make_compatible(*primitive*, *target_version*)

policies

project_id

refresh(**args*, ***kwargs*)
Refreshes the instance group.

save(**args*, ***kwargs*)
Save updates to this instance group.

updated_at

user_id

uuid

class InstanceGroupList(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.6'

child_versions = {'1.4': '1.7', '1.5': '1.8', '1.6': '1.9', '1.0': '1.3', '1.1': '1.4', '1.2': '1.5', '1.3': '1.6'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

classmethod get_by_project_id(*context*, **args*, ***kwargs*)

objects

3.7.507 The nova.objects.instance_info_cache Module

class InstanceInfoCache(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.5'

created_at

delete(**args*, ***kwargs*)

deleted

deleted_at

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'del

classmethod get_by_instance_uuid(*context*, **args*, ***kwargs*)

instance_uuid

network_info

classmethod new(*context*, *instance_uuid*)
Create an InfoCache object that can be used to create the DB entry for the first time.

When save()ing this object, the info_cache_update() DB call will properly handle creating it if it doesn't exist already.

refresh(**args*, ***kwargs*)

save(**args*, ***kwargs*)

updated_at

3.7.508 The nova.objects.instance_mapping Module

class InstanceMapping(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaTimestampObject, nova.objects.base.NovaObject

VERSION = '1.0'

cell_id

create(**args*, ***kwargs*)

created_at

destroy(**args*, ***kwargs*)

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'cel

classmethod get_by_instance_uuid(*context*, **args*, ***kwargs*)

id

instance_uuid

project_id

save(**args*, ***kwargs*)

updated_at

class InstanceMappingList(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.0'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_by_project_id(*context*, **args*, ***kwargs*)

objects

3.7.509 The nova.objects.instance_numa_topology Module

class InstanceNUMACell(***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

cpu_pinning
Descriptor allowing us to assign pinning data as a dict of key_types

This allows us to have an object field that will be a dict of key_type keys, allowing that will convert back to string-keyed dict.

This will take care of the conversion while the dict field will make sure that we store the raw json-serializable data on the object.

key_type should return a type that unambiguously responds to six.text_type so that calling key_type on it yields the same thing.

cpu_pinning_raw

cpu_pinning_requested

cpu_topology

cpuset

fields = {'cpu_topology': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'pag

id

memory

obj_relationships = {'cpu_topology': [('1.2', '1.0')]}

pagesize

pin(*vcpu*, *pcpu*)

pin_vcpus(**cpu_pairs*)

siblings

class InstanceNUMATopology(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

cells

cpu_pinning_requested

create(**args*, ***kwargs*)

classmethod delete_by_instance_uuid(*context*, *instance_uuid*)

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'cel

classmethod get_by_instance_uuid(*context*, **args*, ***kwargs*)

id

instance_uuid

classmethod obj_from_db_obj(*instance_uuid*, *db_obj*)

classmethod obj_from_primitive(*primitive*)

obj_relationships = {'cells': [('1.0', '1.0')]}

3.7.510 The nova.objects.instance_pci_requests Module

class InstancePCIRequest (*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

alias_name

count

fields = {'count': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'is_new': I

is_new

new

obj_load_attr (*attr*)

obj_make_compatible (*primitive*, *target_version*)

request_id

spec

class InstancePCIRequests (*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'req

classmethod from_request_spec_instance_props (*pci_requests*)

classmethod get_by_instance (*context*, *instance*)

classmethod get_by_instance_uuid (*context*, **args*, ***kwargs*)

classmethod get_by_instance_uuid_and_newness (*context*, *instance_uuid*, *is_new*)

instance_uuid

classmethod obj_from_db (*context*, *instance_uuid*, *db_requests*)

obj_make_compatible (*primitive*, *target_version*)

obj_relationships = {'requests': [('1.0', '1.0'), ('1.1', '1.1')]}

requests

save (**args*, ***kwargs*)

to_json ()

3.7.511 The nova.objects.keypair Module

class KeyPair (*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.3'

create (**args*, ***kwargs*)

created_at

deleted

deleted_at

destroy(**args*, ***kwargs*)

classmethod destroy_by_name(*context*, **args*, ***kwargs*)

fields = {'public_key': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'user_i

fingerprint

classmethod get_by_name(*context*, **args*, ***kwargs*)

id

name

obj_make_compatible(*primitive*, *target_version*)

public_key

type

updated_at

user_id

class KeyPairList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.2'

child_versions = {'1.0': '1.1', '1.1': '1.2', '1.2': '1.3'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_by_user(*context*, **args*, ***kwargs*)

classmethod get_count_by_user(*context*, **args*, ***kwargs*)

objects

3.7.512 The nova.objects.migration Module

class Migration(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

dest_compute

dest_host

dest_node

fields = {'status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'instance_u

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_instance_and_status(*context*, **args*, ***kwargs*)

hidden

id

instance

instance_uuid

migration_type

new_instance_type_id

obj_load_attr(*attrname*)

obj_make_compatible(*primitive*, *target_version*)

old_instance_type_id

save(**args*, ***kwargs*)

source_compute

source_node

status

updated_at

class MigrationList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.2'

child_versions = {'1.0': '1.1', '1.1': '1.1', '1.2': '1.2'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_by_filters(*context*, **args*, ***kwargs*)

classmethod get_in_progress_by_host_and_node(*context*, **args*, ***kwargs*)

classmethod get_unconfirmed_by_dest_compute(*context*, **args*, ***kwargs*)

objects

3.7.513 The nova.objects.monitor_metric Module

class MonitorMetric(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject

VERSION = '1.0'

fields = {'timestamp': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'na

name

source

timestamp

to_dict()

value

class MonitorMetricList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.0'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

objects

to_list()

3.7.514 The nova.objects.network Module

class Network(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

classmethod associate(*context*, **args*, ***kwargs*)

bridge

bridge_interface

broadcast

cidr

cidr_v6

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

destroy(**args*, ***kwargs*)

dhcp_server

dhcp_start

classmethod disassociate(*context*, **args*, ***kwargs*)

dns1

dns2

enable_dhcp

fields = {'bridge': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'mtu': Inte

gateway

gateway_v6

classmethod get_by_cidr(*context*, **args*, ***kwargs*)

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

host

id

classmethod in_use_on_host(*context*, **args*, ***kwargs*)

injected

label

mtu

multi_host

netmask

netmask_v6

obj_make_compatible(*primitive*, *target_version*)

priority

project_id

rxtx_base

save(**args*, ***kwargs*)

share_address

updated_at

uuid

vlan

vpn_private_address

vpn_public_address

vpn_public_port

class NetworkList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.2'

child_versions = {'1.0': '1.0', '1.1': '1.1', '1.2': '1.2'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

classmethod get_by_host(*context*, **args*, ***kwargs*)

classmethod get_by_project(*context*, **args*, ***kwargs*)

classmethod get_by_uuids(*context*, **args*, ***kwargs*)

objects

3.7.515 The nova.objects.network_request Module

class NetworkRequest(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

address

fields = {'network_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'pci_r

classmethod from_tuple(*net_tuple*)

network_id

obj_load_attr(*attr*)

pci_request_id

port_id

to_tuple()

class **NetworkRequestList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

as_tuples()

child_versions = {'1.0': '1.0', '1.1': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

is_single_unspecified

objects

3.7.516 The nova.objects.numa Module

class **NUMACell**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

avail_cpus

avail_memory

can_fit_hugepages(*pagesize*, *memory*)

Returns whether memory can fit into hugepages size

Parameters

- **pagesize** – a page size in KibB
- **memory** – a memory size asked to fit in KiB

Returns whether memory can fit in hugepages

Raises MemoryPageSizeNotSupported if page size not supported

cpu_usage

cpuset

fields = {'cpu_usage': Integer(default=0,nullable=False), 'memory_usage': Integer(default=0,nullable=False), 'cpuset':

free_cpus

free_siblings

id

memory

memory_usage

mempages

obj_relationships = {'mempages': [('1.2', '1.0')]}

pin_cpus(*cpus*)

pinned_cpus

siblings

unpin_cpus(*cpus*)

class NUMAPagesTopology(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

fields = {'total': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'size_kb': I

free
Returns the number of avail pages.

free_kb
Returns the avail memory size in KiB.

size_kb

total

used

class NUMATopology(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

cells

fields = {'cells': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod obj_from_db_obj(*db_obj*)

classmethod obj_from_primitive(*primitive*)

obj_relationships = {'cells': [('1.0', '1.0'), ('1.1', '1.1'), ('1.2', '1.2')]}

class NUMATopologyLimits(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject

VERSION = '1.0'

cpu_allocation_ratio

fields = {'ram_allocation_ratio': Float(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=Fals

classmethod obj_from_db_obj(*db_obj*)

ram_allocation_ratio

to_dict_legacy(*host_topology*)

all_things_equal(*obj_a*, *obj_b*)

3.7.517 The nova.objects.pci_device Module

class PciDevice(**args*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

Object to represent a PCI device on a compute node.

PCI devices are managed by the compute resource tracker, which discovers the devices from the hardware platform, claims, allocates and frees devices for instances.

The PCI device information is permanently maintained in a database. This makes it convenient to get PCI device information, like physical function for a VF device, adjacent switch IP address for a NIC, hypervisor identification for a PCI device, etc. It also provides a convenient way to check device allocation information for administrator purposes.

A device can be in available/claimed/allocated/deleted/removed state.

A device is available when it is discovered..

A device is claimed prior to being allocated to an instance. Normally the transition from claimed to allocated is quick. However, during a resize operation the transition can take longer, because devices are claimed in prep_resize and allocated in finish_resize.

A device becomes removed when hot removed from a node (i.e. not found in the next auto-discover) but not yet synced with the DB. A removed device should not be allocated to any instance, and once deleted from the DB, the device object is changed to deleted state and no longer synced with the DB.

Filed notes:

```
| 'dev_id':
|   Hypervisor's identification for the device, the string format
|   is hypervisor specific
| 'extra_info':
|   Device-specific properties like PF address, switch ip address etc.
```

VERSION = '1.3'

address

compute_node_id

classmethod create(*dev_dict*)

Create a PCI device based on hypervisor information.

As the device object is just created and is not synced with db yet thus we should not reset changes here for fields from dict.

created_at

deleted

deleted_at

dev_id

dev_type

extra_info

fields = {'status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'dev_id': St

classmethod get_by_dev_addr(*context*, **args*, ***kwargs*)

classmethod get_by_dev_id(*context*, **args*, ***kwargs*)

id

instance_uuid

label

numa_node

obj_make_compatible(*primitive*, *target_version*)

product_id

request_id

save(**args*, ***kwargs*)

status

update_device(*dev_dict*)
Sync the content from device dictionary to device object.

The resource tracker updates the available devices periodically. To avoid meaningless syncs with the database, we update the device object only if a value changed.

updated_at

vendor_id

class **PciDeviceList**(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

child_versions = {'1.0': '1.1', '1.1': '1.2', '1.2': '1.3'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_by_compute_node**(*context*, **args*, ***kwargs*)

classmethod **get_by_instance_uuid**(*context*, **args*, ***kwargs*)

objects

compare_pci_device_attributes(*obj_a*, *obj_b*)

3.7.518 The nova.objects.pci_device_pool Module

class **PciDevicePool**(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaObject

VERSION = '1.1'

count

fields = {'count': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'numa_no

classmethod **from_dict**(*value*)

numa_node

obj_make_compatible(*primitive*, *target_version*)

product_id

tags

to_dict()

vendor_id

class **PciDevicePoolList**(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

child_versions = {'1.0': '1.0', '1.1': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

objects

from_pci_stats(*pci_stats*)
Create and return a PciDevicePoolList from the data stored in the db, which can be either the serialized object, or, prior to the creation of the device pool objects, a simple dict or a list of such dicts.

3.7.519 The nova.objects.quotas Module

class Quotas(**args*, ***kwargs*)
Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.2'

commit(**args*, ***kwargs*)

classmethod count(*context*, **args*, ***kwargs*)
Count a resource.

classmethod create_limit(*context*, **args*, ***kwargs*)

fields = {'reservations': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project

classmethod from_reservations(*context*, *reservations*, *instance=None*)
Transitional for compatibility.

classmethod limit_check(*context*, **args*, ***kwargs*)
Check quota limits.

project_id

reservations

reserve(**args*, ***kwargs*)

rollback(**args*, ***kwargs*)
Rollback quotas.

classmethod update_limit(*context*, **args*, ***kwargs*)

user_id

class QuotasNoOp(**args*, ***kwargs*)
Bases: nova.objects.quotas.Quotas

commit(*context=None*)

fields = {'reservations': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project

project_id

reservations

reserve(*context*, *expire=None*, *project_id=None*, *user_id=None*, ***deltas*)

rollback(*context=None*)

user_id

ids_from_instance(*context*, *instance*)

ids_from_security_group(*context*, *security_group*)

ids_from_server_group(*context*, *server_group*)

3.7.520 The nova.objects.security_group Module

class SecurityGroup(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

created_at

deleted

deleted_at

description

fields = {'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'use

classmethod get(*context*, **args*, ***kwargs*)

classmethod get_by_name(*context*, **args*, ***kwargs*)

id

in_use(**args*, ***kwargs*)

name

project_id

refresh(**args*, ***kwargs*)

save(**args*, ***kwargs*)

updated_at

user_id

class SecurityGroupList(**args*, ***kwargs*)
Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

classmethod get_by_instance(*context*, **args*, ***kwargs*)

classmethod get_by_project(*context*, **args*, ***kwargs*)

objects

make_secgroup_list(*security_groups*)
A helper to make security group objects from a list of names.

Note that this does not make them save-able or have the rest of the attributes they would normally have, but provides a quick way to fill, for example, an instance object during create.

3.7.521 The nova.objects.security_group_rule Module

class SecurityGroupRule(*context=None*, ***kwargs*)
Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.1'

cidr

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

fields = {'from_port': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'proto

from_port

classmethod **get_by_id**(*context*, **args*, ***kwargs*)

grantee_group

id

obj_relationships = {'parent_group': [('1.0', '1.1'), ('1.1', '1.1')], 'grantee_group': [('1.0', '1.1'), ('1.1', '1.1')]}

parent_group

protocol

to_port

updated_at

class **SecurityGroupRuleList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

child_versions = {'1.0': '1.0', '1.1': '1.1'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_by_security_group**(*context*, *security_group*)

classmethod **get_by_security_group_id**(*context*, **args*, ***kwargs*)

objects

3.7.522 The nova.objects.service Module

class **Service**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.13'

availability_zone

binary

compute_node

create(**args*, ***kwargs*)

created_at

deleted

deleted_at

destroy(**args*, ***kwargs*)

disabled

disabled_reason

fields = {'binary': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'compute_

classmethod **get_by_args**(*context*, **args*, ***kwargs*)

classmethod **get_by_compute_host**(*context*, **args*, ***kwargs*)

classmethod **get_by_host_and_binary**(*context*, **args*, ***kwargs*)

classmethod **get_by_host_and_topic**(*context*, **args*, ***kwargs*)

classmethod **get_by_id**(*context*, **args*, ***kwargs*)

host

id

last_seen_up

obj_load_attr(*attrname*)

obj_make_compatible(*primitive*, *target_version*)

obj_relationships = {'compute_node': [('1.1', '1.4'), ('1.3', '1.5'), ('1.5', '1.6'), ('1.7', '1.8'), ('1.8', '1.9'), ('1.9', '1.10

report_count

save(**args*, ***kwargs*)

topic

updated_at

class **ServiceList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.11'

child_versions = {'1.4': '1.6', '1.5': '1.7', '1.6': '1.8', '1.7': '1.9', '1.0': '1.2', '1.1': '1.3', '1.2': '1.4', '1.3': '1.5', '1.8'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_all**(*context*, **args*, ***kwargs*)

classmethod **get_by_binary**(*context*, **args*, ***kwargs*)

classmethod **get_by_host**(*context*, **args*, ***kwargs*)

classmethod **get_by_topic**(*context*, **args*, ***kwargs*)

objects

3.7.523 The nova.objects.tag Module

class **Tag**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject

VERSION = '1.1'

create(**args*, ***kwargs*)

classmethod **destroy**(*context*, **args*, ***kwargs*)

classmethod **exists**(*context*, **args*, ***kwargs*)

fields = {'tag': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'resource_id':

resource_id

tag

class **TagList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.1'

child_versions = {'1.0': '1.0', '1.1': '1.1'}

classmethod **create**(*context*, **args*, ***kwargs*)

classmethod **destroy**(*context*, **args*, ***kwargs*)

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_by_resource_id**(*context*, **args*, ***kwargs*)

objects

3.7.524 The nova.objects.task_log Module

class **TaskLog**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject

VERSION = '1.0'

begin_task(**args*, ***kwargs*)

created_at

deleted

deleted_at

end_task(**args*, ***kwargs*)

errors

fields = {'errors': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'task_nam

classmethod **get**(*obj*, **args*, ***kwargs*)

host

id

message

period_beginning

period_ending

state

task_items

task_name

updated_at

class **TaskLogList**(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.0'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod **get_all**(*obj*, **args*, ***kwargs*)

objects

3.7.525 The nova.objects.vcpu_model Module

class **VirtCPUFeature**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject

VERSION = '1.0'

fields = {'policy': CPUFeaturePolicy(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True,v

name

obj_load_attr(*attrname*)

policy

class **VirtCPUModel**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject

VERSION = '1.0'

arch

features

fields = {'vendor': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'features':

classmethod **from_json**(*jsonstr*)

classmethod **get_by_instance_uuid**(*context*, **args*, ***kwargs*)

match

mode

model

obj_load_attr(*attrname*)

obj_relationships = {'features': [('1.0', '1.0')], 'topology': [('1.0', '1.0')]}

to_json()

topology

vendor

3.7.526 The nova.objects.virt_cpu_topology Module

class **VirtCPUTopology**(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

cores

fields = {'cores': Integer(default=1,nullable=True), 'threads': Integer(default=1,nullable=True), 'sockets': Integer(defa

classmethod **from_dict**(*data*)

sockets

threads

to_dict()

3.7.527 The nova.objects.virtual_interface Module

class VirtualInterface(*context=None*, ***kwargs*)

Bases: nova.objects.base.NovaPersistentObject, nova.objects.base.NovaObject, nova.objects.base.NovaObjectDictCompat

VERSION = '1.0'

address

create(**args*, ***kwargs*)

created_at

classmethod delete_by_instance_uuid(*context*, **args*, ***kwargs*)

deleted

deleted_at

fields = {'instance_uuid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'uu

classmethod get_by_address(*context*, **args*, ***kwargs*)

classmethod get_by_id(*context*, **args*, ***kwargs*)

classmethod get_by_instance_and_network(*context*, **args*, ***kwargs*)

classmethod get_by_uuid(*context*, **args*, ***kwargs*)

id

instance_uuid

network_id

updated_at

uuid

class VirtualInterfaceList(**args*, ***kwargs*)

Bases: nova.objects.base.ObjectListBase, nova.objects.base.NovaObject

VERSION = '1.0'

child_versions = {'1.0': '1.0'}

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

classmethod get_by_instance_uuid(*context*, **args*, ***kwargs*)

objects

3.7.528 The nova.objectstore.s3server Module

Implementation of an S3-like storage server based on local files.

Useful to test features that will eventually run on S3, or if you want to run something locally that was once running on S3.

We don't support all the features of S3, but it does work with the standard S3 client for the most basic semantics. To use the standard S3 client with this module:

```
c = S3.AWSAuthConnection("", "", server="localhost", port=8888,
                         is_secure=False)
c.create_bucket("mybucket")
c.put("mybucket", "mykey", "a value")
print c.get("mybucket", "mykey").body
```

class BaseRequestHandler(*application*)

Bases: object

Base class emulating Tornado's web framework pattern in WSGI.

This is a direct port of Tornado's implementation, so some key decisions about how the code interacts have already been chosen.

The two most common ways of designing web frameworks can be classified as async object-oriented and sync functional.

Tornado's is on the OO side because a response is built up in and using the shared state of an object and one of the object's methods will eventually trigger the "finishing" of the response asynchronously.

Most WSGI stuff is in the functional side, we pass a request object to every call down a chain and the eventual return value will be a response.

Part of the function of the routing code in S3Application as well as the code in BaseRequestHandler's __call__ method is to merge those two styles together enough that the Tornado code can work without extensive modifications.

To do that it needs to give the Tornado-style code clean objects that it can modify the state of for each request that is processed, so we use a very simple factory lambda to create new state for each request, that's the stuff in the router, and when we let the Tornado code modify that object to handle the request, then we return the response it generated. This wouldn't work the same if Tornado was being more async'y and doing other callbacks throughout the process, but since Tornado is being relatively simple here we can be satisfied that the response will be complete by the end of the get/post method.

finish(*body=''*)

get_argument(*arg*, *default*)

invalid(***kwargs*)

render_xml(*value*)

set_404()

set_header(*header*, *value*)

set_status(*status_code*)

class BucketHandler(*application*)

Bases: nova.objectstore.s3server.BaseRequestHandler

delete(*bucket_name*)

get(*bucket_name*)

head(*bucket_name*)

put(*bucket_name*)

class ObjectHandler(*application*)
Bases: nova.objectstore.s3server.BaseRequestHandler

delete(*bucket*, *object_name*)

get(*bucket*, *object_name*)

put(*bucket*, *object_name*)

class RootHandler(*application*)
Bases: nova.objectstore.s3server.BaseRequestHandler

get()

class S3Application(*root_directory*, *bucket_depth=0*, *mapper=None*)
Bases: nova.wsgi.Router

Implementation of an S3-like storage server based on local files.

If bucket depth is given, we break files up into multiple directories to prevent hitting file system limits for number of files in each directories. 1 means one level of directories, 2 means 2, etc.

get_wsgi_server()

3.7.529 The nova.openstack.common._i18n Module

oslo.i18n integration module.

See http://docs.openstack.org/developer/oslo.i18n/usage.html

3.7.530 The nova.openstack.common.cliutils Module

exception MissingArgs(*missing*)
Bases: exceptions.Exception

Supplied arguments are not sufficient for calling a function.

add_arg(*func*, **args*, ***kwargs*)
Bind CLI arguments to a shell.py *do_foo* function.

arg(**args*, ***kwargs*)
Decorator for CLI args.

Example:

```
>>> @arg("name", help="Name of the new entity")
... def entity_create(args):
...     pass
```

env(**args*, ***kwargs*)
Returns the first environment variable set.

If all are empty, defaults to '' or keyword arg *default*.

exit(*msg=''*)

get_password(*max_password_prompts=3*)
Read password from TTY.

get_service_type(*f*)
Retrieves service type from function.

isunauthenticated(*func*)
Checks if the function does not require authentication.

Mark such functions with the *@unauthenticated* decorator.

Returns bool

pretty_choice_list(*l*)

print_dict(*dct*, *dict_property='Property'*, *wrap=0*)
Print a *dict* as a table of two columns.

Parameters

- **dct** – *dict* to print
- **dict_property** – name of the first column
- **wrap** – wrapping for the second column

print_list(*objs*, *fields*, *formatters=None*, *sortby_index=0*, *mixed_case_fields=None*, *field_labels=None*)
Print a list or objects as a table, one row per object.

Parameters

- **objs** – iterable of `Resource`
- **fields** – attributes that correspond to columns, in order
- **formatters** – *dict* of callables for field formatting
- **sortby_index** – index of the field for sorting table rows
- **mixed_case_fields** – fields corresponding to object attributes that have mixed case names (e.g., 'serverId')
- **field_labels** – Labels to use in the heading of the table, default to fields.

service_type(*stype*)
Adds 'service_type' attribute to decorated function.

Usage:

```
@service_type('volume')
def mymethod(f):
...
```

unauthenticated(*func*)
Adds 'unauthenticated' attribute to decorated function.

Usage:

```
>>> @unauthenticated
... def mymethod(f):
...     pass
```

validate_args(*fn*, **args*, ***kwargs*)
Check that the supplied args are sufficient for calling a function.

```
>>> validate_args(lambda a: None)
Traceback (most recent call last):
    ...
MissingArgs: Missing argument(s): a
```

```
>>> validate_args(lambda a, b, c, d: None, 0, c=1)
Traceback (most recent call last):
    ...
MissingArgs: Missing argument(s): b, d
```

Parameters

- **fn** – the function to check
- **arg** – the positional arguments supplied
- **kwargs** – the keyword arguments supplied

3.7.531 The nova.openstack.common.eventlet_backdoor Module

exception EventletBackdoorConfigValueError(*port_range*, *help_msg*, *ex*)
Bases: exceptions.Exception

initialize_if_enabled()

list_opts()
Entry point for oslo-config-generator.

3.7.532 The nova.openstack.common.fileutils Module

delete_cached_file(*filename*)
Delete cached file if present.

Parameters **filename** – filename to delete

delete_if_exists(*path*, *remove=<built-in function unlink>*)
Delete a file, but ignore file not found error.

Parameters

- **path** – File to delete
- **remove** – Optional function to remove passed path

ensure_tree(*path*, *mode=511*)
Create a directory (and any ancestor directories required)

Parameters

- **path** – Directory to create
- **mode** – Directory creation permissions

file_open(**args*, ***kwargs*)
Open file

see built-in open() documentation for more details

Note: The reason this is kept in a separate module is to easily be able to provide a stub module that doesn't alter system state at all (for unit tests)

read_cached_file(*filename*, *force_reload=False*)
Read from a file if it has been modified.

Parameters **force_reload** – Whether to reload the file.

Returns A tuple with a boolean specifying if the data is fresh or not.

remove_path_on_error(**args*, ***kwds*)
Protect code that wants to operate on PATH atomically. Any exception will cause PATH to be removed.

Parameters

- **path** – File to work with
- **remove** – Optional function to remove passed path

write_to_tempfile(*content*, *path=None*, *suffix=''*, *prefix='tmp'*)
Create temporary file or use existing file.

This util is needed for creating temporary file with specified content, suffix and prefix. If path is not None, it will be used for writing content. If the path doesn't exist it'll be created.

Parameters

- **content** – content for temporary file.
- **path** – same as parameter 'dir' for mkstemp
- **suffix** – same as parameter 'suffix' for mkstemp
- **prefix** – same as parameter 'prefix' for mkstemp

For example: it can be used in database tests for creating configuration files.

3.7.533 The nova.openstack.common.imageutils Module

Helper methods to deal with images.

class QemuImgInfo(*cmd_output=None*)
Bases: object

BACKING_FILE_RE = <_sre.SRE_Pattern object at 0x7f8590f83830>

SIZE_RE = <_sre.SRE_Pattern object at 0x7f859bbb7590>

TOP_LEVEL_RE = <_sre.SRE_Pattern object at 0x7f8581782c90>

3.7.534 The nova.openstack.common.loopingcall Module

class DynamicLoopingCall(*f=None*, **args*, ***kw*)
Bases: nova.openstack.common.loopingcall.LoopingCallBase

A looping call which sleeps until the next known event.

The function called should return how long to sleep for before being called again.

start(*initial_delay=None*, *periodic_interval_max=None*)

class FixedIntervalLoopingCall(*f=None*, **args*, ***kw*)
Bases: nova.openstack.common.loopingcall.LoopingCallBase

A fixed interval looping call.

start(*interval*, *initial_delay=None*)

class LoopingCallBase(*f=None*, **args*, ***kw*)
Bases: object

stop()

wait()

exception LoopingCallDone(*retvalue=True*)
Bases: exceptions.Exception

Exception to break out and stop a LoopingCallBase.

The poll-function passed to LoopingCallBase can raise this exception to break out of the loop normally. This is somewhat analogous to StopIteration.

An optional return-value can be included as the argument to the exception; this return-value will be returned by LoopingCallBase.wait()

3.7.535 The nova.openstack.common.memorycache Module

Super simple fake memcache client.

class Client(**args*, ***kwargs*)
Bases: object

Replicates a tiny subset of memcached client interface.

add(*key*, *value*, *time=0*, *min_compress_len=0*)
Sets the value for a key if it doesn't exist.

delete(*key*, *time=0*)
Deletes the value associated with a key.

get(*key*)
Retrieves the value for a key or None.

This expunges expired keys during each get.

incr(*key*, *delta=1*)
Increments the value for a key.

set(*key*, *value*, *time=0*, *min_compress_len=0*)
Sets the value for a key.

get_client(*memcached_servers=None*)

list_opts()
Entry point for oslo-config-generator.

3.7.536 The nova.openstack.common.periodic_task Module

exception InvalidPeriodicTaskArg
Bases: exceptions.Exception

message = u'Unexpected argument for periodic task creation: %(arg)s.'

class PeriodicTasks
Bases: object

add_periodic_task(*task*)
Add a periodic task to the list of periodic tasks.

The task should already be decorated by @periodic_task.

run_periodic_tasks(*context*, *raise_on_error=False*)
Tasks to be run at a periodic interval.

list_opts()
Entry point for oslo-config-generator.

periodic_task(**args*, ***kwargs*)

Decorator to indicate that a method is a periodic task.

This decorator can be used in two ways:

1. Without arguments '@periodic_task', this will be run on the default interval of 60 seconds.
2. With arguments: @periodic_task(spacing=N [, run_immediately=[True|False]] [, name=[None|"string"]) this will be run on approximately every N seconds. If this number is negative the periodic task will be disabled. If the run_immediately argument is provided and has a value of 'True', the first run of the task will be shortly after task scheduler starts. If run_immediately is omitted or set to 'False', the first time the task runs will be approximately N seconds after the task scheduler starts. If name is not provided, __name__ of function is used.

3.7.537 The nova.openstack.common.policy Module

Common Policy Engine Implementation

Policies can be expressed in one of two forms: A list of lists, or a string written in the new policy language.

In the list-of-lists representation, each check inside the innermost list is combined as with an "and" conjunction– for that check to pass, all the specified checks must pass. These innermost lists are then combined as with an "or" conjunction. As an example, take the following rule, expressed in the list-of-lists representation:

```
[["role:admin"], ["project_id:%(project_id)s", "role:projectadmin"]]
```

This is the original way of expressing policies, but there now exists a new way: the policy language.

In the policy language, each check is specified the same way as in the list-of-lists representation: a simple "a:b" pair that is matched to the correct class to perform that check:

```
+=========================================================================+
|            TYPE               |                SYNTAX                   |
+=========================================================================+
|User's Role                    |              role:admin                 |
+-------------------------------+-----------------------------------------+
|Rules already defined on policy|          rule:admin_required            |
+-------------------------------+-----------------------------------------+
|Against URL's[1]               |         http://my-url.org/check         |
+-------------------------------+-----------------------------------------+
|User attributes[2]             |    project_id:%(target.project.id)s     |
+-------------------------------+-----------------------------------------+
|Strings                        |        <variable>:'xpto2035abc'         |
|                               |         'myproject':<variable>          |
+-------------------------------+-----------------------------------------+
|                               |         project_id:xpto2035abc          |
|Literals                       |         domain_id:20                    |
|                               |         True:%(user.enabled)s           |
+=========================================================================+
```

[1]URL checking must return 'True' to be valid [2]User attributes (obtained through the token): user_id, domain_id or project_id

Conjunction operators are available, allowing for more expressiveness in crafting policies. So, in the policy language, the previous check in list-of-lists becomes:

```
role:admin or (project_id:%(project_id)s and role:projectadmin)
```

The policy language also has the "not" operator, allowing a richer policy rule:

```
project_id:%(project_id)s and not role:dunce
```

Attributes sent along with API calls can be used by the policy engine (on the right side of the expression), by using the following syntax:

```
<some_value>:%(user.id)s
```

Contextual attributes of objects identified by their IDs are loaded from the database. They are also available to the policy engine and can be checked through the *target* keyword:

```
<some_value>:%(target.role.name)s
```

Finally, two special policy checks should be mentioned; the policy check "@" will always accept an access, and the policy check "!" will always reject an access. (Note that if a rule is either the empty list ("[]") or the empty string, this is equivalent to the "@" policy check.) Of these, the "!" policy check is probably the most useful, as it allows particular rules to be explicitly disabled.

class AndCheck(*rules*)
Bases: nova.openstack.common.policy.BaseCheck

Implements the "and" logical operator.

A policy check that requires that a list of other checks all return True.

add_check(*rule*)
Adds rule to be tested.

Allows addition of another rule to the list of rules that will be tested. Returns the AndCheck object for convenience.

class BaseCheck
Bases: object

Abstract base class for Check classes.

class Check(*kind*, *match*)
Bases: nova.openstack.common.policy.BaseCheck

A base class to allow for user-defined policy checks.

class Enforcer(*policy_file=None*, *rules=None*, *default_rule=None*, *use_conf=True*, *overwrite=True*)
Bases: object

Responsible for loading and enforcing rules.

Parameters

- **policy_file** – Custom policy file to use, if none is specified, *CONF.policy_file* will be used.
- **rules** – Default dictionary / Rules to use. It will be considered just in the first instantiation. If *load_rules(True)*, *clear()* or *set_rules(True)* is called this will be overwritten.
- **default_rule** – Default rule to use, CONF.default_rule will be used if none is specified.
- **use_conf** – Whether to load rules from cache or config file.
- **overwrite** – Whether to overwrite existing rules when reload rules from config file.

clear()
Clears Enforcer rules, policy's cache and policy's path.

enforce(*rule*, *target*, *creds*, *do_raise=False*, *exc=None*, **args*, ***kwargs*)
Checks authorization of a rule against the target and credentials.

Parameters

- **rule** – A string or BaseCheck instance specifying the rule to evaluate.
- **target** – As much information about the object being operated on as possible, as a dictionary.
- **creds** – As much information about the user performing the action as possible, as a dictionary.
- **do_raise** – Whether to raise an exception or not if check fails.
- **exc** – Class of the exception to raise if the check fails. Any remaining arguments passed to enforce() (both positional and keyword arguments) will be passed to the exception class. If not specified, PolicyNotAuthorized will be used.

Returns Returns False if the policy does not allow the action and exc is not provided; otherwise, returns a value that evaluates to True. Note: for rules using the "case" expression, this True value will be the specified string from the expression.

load_rules(*force_reload=False*)

Loads policy_path's rules.

Policy file is cached and will be reloaded if modified.

Parameters **force_reload** – Whether to reload rules from config file.

set_rules(*rules*, *overwrite=True*, *use_conf=False*)

Create a new Rules object based on the provided dict of rules.

Parameters

- **rules** – New rules to use. It should be an instance of dict.
- **overwrite** – Whether to overwrite current rules or update them with the new rules.
- **use_conf** – Whether to reload rules from cache or config file.

class FalseCheck

Bases: nova.openstack.common.policy.BaseCheck

A policy check that always returns False (disallow).

class GenericCheck(*kind*, *match*)

Bases: nova.openstack.common.policy.Check

class HttpCheck(*kind*, *match*)

Bases: nova.openstack.common.policy.Check

class NotCheck(*rule*)

Bases: nova.openstack.common.policy.BaseCheck

Implements the "not" logical operator.

A policy check that inverts the result of another policy check.

class OrCheck(*rules*)

Bases: nova.openstack.common.policy.BaseCheck

Implements the "or" operator.

A policy check that requires that at least one of a list of other checks returns True.

add_check(*rule*)

Adds rule to be tested.

Allows addition of another rule to the list of rules that will be tested. Returns the OrCheck object for convenience.

class ParseState
Bases: object

Implement the core of parsing the policy language.

Uses a greedy reduction algorithm to reduce a sequence of tokens into a single terminal, the value of which will be the root of the Check tree.

Note: error reporting is rather lacking. The best we can get with this parser formulation is an overall "parse failed" error. Fortunately, the policy language is simple enough that this shouldn't be that big a problem.

reduce()
Perform a greedy reduction of the token stream.

If a reducer method matches, it will be executed, then the reduce() method will be called recursively to search for any more possible reductions.

reducers = [(['check', 'or', 'check'], '_make_or_expr'), (['or_expr', 'or', 'check'], '_extend_or_expr'), (['not', 'check'],

result
Obtain the final result of the parse.

Raises ValueError if the parse failed to reduce to a single result.

shift(*tok*, *value*)
Adds one more token to the state. Calls reduce().

class ParseStateMeta
Bases: type

Metaclass for the ParseState class.

Facilitates identifying reduction methods.

exception PolicyNotAuthorized(*rule*)
Bases: exceptions.Exception

class RoleCheck(*kind*, *match*)
Bases: nova.openstack.common.policy.Check

class RuleCheck(*kind*, *match*)
Bases: nova.openstack.common.policy.Check

class Rules(*rules=None*, *default_rule=None*)
Bases: dict

A store for rules. Handles the default_rule setting directly.

classmethod load_json(*data*, *default_rule=None*)
Allow loading of JSON rule data.

class TrueCheck
Bases: nova.openstack.common.policy.BaseCheck

A policy check that always returns True (allow).

list_opts()
Entry point for oslo-config-generator.

parse_rule(*rule*)
Parses a policy rule into a tree of Check objects.

reducer(**tokens*)
Decorator for reduction methods.

Arguments are a sequence of tokens, in order, which should trigger running this reduction method.

register(*name*, *func=None*)
Register a function or Check class as a policy check.

Parameters

- **name** – Gives the name of the check type, e.g., 'rule', 'role', etc. If name is None, a default check type will be registered.
- **func** – If given, provides the function or class to register. If not given, returns a function taking one argument to specify the function or class to register, allowing use as a decorator.

3.7.538 The nova.openstack.common.report.generators.conf Module

Provides OpenStack config generators

This module defines a class for configuration generators for generating the model in openstack.common.report.models.conf.

class ConfigReportGenerator(*cnf=<oslo_config.cfg.ConfigOpts object at 0x7f858e621d90>*)
Bases: object

A Configuration Data Generator

This generator returns openstack.common.report.models.conf.ConfigModel, by default using the configuration options stored in oslo_config.cfg.CONF, which is where OpenStack stores everything.

Parameters **cnf** (oslo_config.cfg.ConfigOpts) – the configuration option object

3.7.539 The nova.openstack.common.report.generators.process Module

Provides process-data generators

This modules defines a class for generating process data by way of the psutil package.

class ProcessReportGenerator
Bases: object

A Process Data Generator

This generator returns a openstack.common.report.models.process.ProcessModel based on the current process (which will also include all subprocesses, recursively) using the psutil.Process class'.

3.7.540 The nova.openstack.common.report.generators.threading Module

Provides thread-related generators

This module defines classes for threading-related generators for generating the models in openstack.common.report.models.threading.

class GreenThreadReportGenerator
Bases: object

A Green Thread Data Generator

This generator returns a collection of openstack.common.report.models.threading.GreenThreadModel objects by introspecting the current python garbage collection state, and sifting through for greenlet.greenlet objects.

See also:

Function `openstack.common.report.utils._find_objects()`

class **ThreadReportGenerator**(*curr_thread_traceback=None*)

Bases: `object`

A Thread Data Generator

This generator returns a collection of `openstack.common.report.models.threading.ThreadModel` objects by introspecting the current python state using `sys._current_frames()` . Its constructor may optionally be passed a frame object. This frame object will be interpreted as the actual stack trace for the current thread, and, come generation time, will be used to replace the stack trace of the thread in which this code is running.

3.7.541 The `nova.openstack.common.report.generators.version` Module

Provides OpenStack version generators

This module defines a class for OpenStack version and package information generators for generating the model in `openstack.common.report.models.version`.

class **PackageReportGenerator**(*version_obj*)

Bases: `object`

A Package Information Data Generator

This generator returns `openstack.common.report.models.version.PackageModel`, extracting data from the given version object, which should follow the general format defined in Nova's version information (i.e. it should contain the methods vendor_string, product_string, and version_string_with_package).

Parameters **version_object** – the version information object

3.7.542 The `nova.openstack.common.report.guru_meditation_report` Module

Provides Guru Meditation Report

This module defines the actual OpenStack Guru Meditation Report class.

This can be used in the OpenStack command definition files. For example, in a nova command module (under nova/cmd):

```
CONF = cfg.CONF
# maybe import some options here...

def main():
    config.parse_args(sys.argv)
    logging.setup('blah')

    TextGuruMeditation.register_section('Some Special Section',
                                        special_section_generator)
    TextGuruMeditation.setup_autorun(version_object)

    server = service.Service.create(binary='some-service',
                                    topic=CONF.some_service_topic)
    service.serve(server)
    service.wait()
```

Then, you can do

```
$ kill -USR1 $SERVICE_PID
```

and get a Guru Meditation Report in the file or terminal where stderr is logged for that given service.

class GuruMeditation(*version_obj*, *sig_handler_tb=None*, **args*, ***kwargs*)
Bases: object

A Guru Meditation Report Mixin/Base Class

This class is a base class for Guru Meditation Reports. It provides facilities for registering sections and setting up functionality to auto-run the report on a certain signal.

This class should always be used in conjunction with a Report class via multiple inheritance. It should always come first in the class list to ensure the MRO is correct.

classmethod handle_signal(*version*, *service_name*, *log_dir*, *traceback*)
The Signal Handler

This method (indirectly) handles receiving a registered signal and dumping the Guru Meditation Report to stderr or a file in a given dir. If service name and log dir are not None, the report will be dumped to a file named $service_name_gurumeditation_$current_time in the log_dir directory. This method is designed to be curried into a proper signal handler by currying out the version parameter.

Parameters

- **version** – the version object for the current product
- **service_name** – this program name used to construct logfile name
- **logdir** – path to a log directory where to create a file
- **traceback** – the traceback provided to the signal handler

classmethod register_section(*section_title*, *generator*)
Register a New Section

This method registers a persistent section for the current class.

Parameters

- **section_title** (*str*) – the title of the section
- **generator** – the generator for the section

run()

classmethod setup_autorun(*version*, *service_name=None*, *log_dir=None*, *signum=None*)
Set Up Auto-Run

This method sets up the Guru Meditation Report to automatically get dumped to stderr or a file in a given dir when the given signal is received.

Parameters

- **version** – the version object for the current product
- **service_name** – this program name used to construct logfile name
- **logdir** – path to a log directory where to create a file
- **signum** – the signal to associate with running the report

timestamp_fmt = '%Y%m%d%H%M%S'

class TextGuruMeditation(*version_obj*, *traceback=None*)
Bases: nova.openstack.common.report.guru_meditation_report.GuruMeditation, nova.openstack.common.report.report.TextReport

A Text Guru Meditation Report

This report is the basic human-readable Guru Meditation Report

It contains the following sections by default (in addition to any registered persistent sections):

- Package Information
- Threads List
- Green Threads List
- Process List
- Configuration Options

Parameters

- **version_obj** – the version object for the current product
- **traceback** – an (optional) frame object providing the actual traceback for the current thread

3.7.543 The `nova.openstack.common.report.models.base` Module

Provides the base report model

This module defines a class representing the basic report data model from which all data models should inherit (or at least implement similar functionality). Data models store unserialized data generated by generators during the report serialization process.

class `ReportModel` (*data=None*, *attached_view=None*)

Bases: `_abcoll.MutableMapping`

A Report Data Model

A report data model contains data generated by some generator method or class. Data may be read or written using dictionary-style access, and may be read (but not written) using object-member-style access. Additionally, a data model may have an associated view. This view is used to serialize the model when str() is called on the model. An appropriate object for a view is callable with a single parameter: the model to be serialized.

If present, the object passed in as data will be transformed into a standard python dict. For mappings, this is fairly straightforward. For sequences, the indices become keys and the items become values.

Parameters

- **data** – a sequence or mapping of data to associate with the model
- **attached_view** – a view object to attach to this model

`set_current_view_type` (*tp*, *visited=None*)

Set the current view type

This method attempts to set the current view type for this model and all submodels by calling itself recursively on all values, traversing intervening sequences and mappings when possible, and ignoring all other objects.

Parameters

- **tp** – the type of the view ('text', 'json', 'xml', etc)
- **visited** – a set of object ids for which the corresponding objects have already had their view type set

3.7.544 The nova.openstack.common.report.models.conf Module

Provides OpenStack Configuration Model

This module defines a class representing the data model for oslo_config configuration options

class ConfigModel (*conf_obj*)

Bases: nova.openstack.common.report.models.with_default_views.ModelWithDefaultViews

A Configuration Options Model

This model holds data about a set of configuration options from oslo_config. It supports both the default group of options and named option groups.

Parameters **conf_obj** (oslo_config.cfg.ConfigOpts) – a configuration object

3.7.545 The nova.openstack.common.report.models.process Module

Provides a process model

This module defines a class representing a process, potentially with subprocesses.

class ProcessModel (*process*)

Bases: nova.openstack.common.report.models.with_default_views.ModelWithDefaultViews

A Process Model

This model holds data about a process, including references to any subprocesses

Parameters **process** – a psutil.Process object

3.7.546 The nova.openstack.common.report.models.threading Module

Provides threading and stack-trace models

This module defines classes representing thread, green thread, and stack trace data models

class GreenThreadModel (*stack*)

Bases: nova.openstack.common.report.models.with_default_views.ModelWithDefaultViews

A Green Thread Model

This model holds data for information about an individual thread. Unlike the thread model, it holds just a stack trace, since green threads do not have thread ids.

See also:

Class StackTraceModel

Parameters **stack** – the python stack state for the green thread

class StackTraceModel (*stack_state*)

Bases: nova.openstack.common.report.models.with_default_views.ModelWithDefaultViews

A Stack Trace Model

This model holds data from a python stack trace, commonly extracted from running thread information

Parameters **stack_state** – the python stack_state object

class ThreadModel (*thread_id*, *stack*)

Bases: nova.openstack.common.report.models.with_default_views.ModelWithDefaultViews

A Thread Model

This model holds data for information about an individual thread. It holds both a thread id, as well as a stack trace for the thread

See also:

Class StackTraceModel

Parameters

- **thread_id** (*int*) – the id of the thread
- **stack** – the python stack state for the current thread

3.7.547 The nova.openstack.common.report.models.version Module

Provides OpenStack Version Info Model

This module defines a class representing the data model for OpenStack package and version information

class PackageModel (*vendor*, *product*, *version*)

Bases: nova.openstack.common.report.models.with_default_views.ModelWithDefaultViews

A Package Information Model

This model holds information about the current package. It contains vendor, product, and version information.

Parameters

- **vendor** (*str*) – the product vendor
- **product** (*str*) – the product name
- **version** (*str*) – the product version

3.7.548 The nova.openstack.common.report.models.with_default_views Module

class ModelWithDefaultViews (**args*, ***kwargs*)

Bases: nova.openstack.common.report.models.base.ReportModel

A Model With Default Views of Various Types

A model with default views has several predefined views, each associated with a given type. This is often used for when a submodel should have an attached view, but the view differs depending on the serialization format

Parameters are as the superclass, except for any parameters ending in '_view': these parameters get stored as default views.

The default 'default views' are

text `openstack.common.report.views.text.generic.KeyValueView`

xml `openstack.common.report.views.xml.generic.KeyValueView`

json `openstack.common.report.views.json.generic.KeyValueView`

to_type()
('type' is one of the 'default views' defined for this model) Serializes this model using the default view for 'type'

Return type str

Returns this model serialized as 'type'

set_current_view_type(*tp*, *visited=None*)

3.7.549 The nova.openstack.common.report.report Module

Provides Report classes

This module defines various classes representing reports and report sections. All reports take the form of a report class containing various report sections.

class BasicReport
Bases: object

A Basic Report

A Basic Report consists of a collection of ReportSection objects, each of which contains a top-level model and generator. It collects these sections into a cohesive report which may then be serialized by calling run().

add_section(*view*, *generator*, *index=None*)
Add a section to the report

This method adds a section with the given view and generator to the report. An index may be specified to insert the section at a given location in the list; If no index is specified, the section is appended to the list. The view is called on the model which results from the generator when the report is run. A generator is simply a method or callable object which takes no arguments and returns a openstack.common.report.models.base.ReportModel or similar object.

Parameters

- **view** – the top-level view for the section
- **generator** – the method or class which generates the model
- **index** (*int or None*) – the index at which to insert the section (or None to append it)

run()
Run the report

This method runs the report, having each section generate its data and serialize itself before joining the sections together. The BasicReport accomplishes the joining by joining the serialized sections together with newlines.

Return type str

Returns the serialized report

class ReportOfType(*tp*)
Bases: nova.openstack.common.report.report.BasicReport

A Report of a Certain Type

A ReportOfType has a predefined type associated with it. This type is automatically propagated down to the each of the sections upon serialization by wrapping the generator for each section.

See also:

Class openstack.common.report.models.with_default_view.ModelWithDefaultView # noqa
(the entire class)

Class openstack.common.report.models.base.ReportModel openstack.common.report.models.bas
noqa

Parameters **tp** (*str*) – the type of the report

add_section(*view*, *generator*, *index=None*)

class ReportSection(*view*, *generator*)
Bases: object

A Report Section

A report section contains a generator and a top-level view. When something attempts to serialize the section by calling str() on it, the section runs the generator and calls the view on the resulting model.

See also:

Class BasicReport BasicReport.add_section()

Parameters

- **view** – the top-level view for this section
- **generator** – the generator for this section (any callable object which takes no parameters and returns a data model)

class TextReport(*name*)
Bases: nova.openstack.common.report.report.ReportOfType

A Human-Readable Text Report

This class defines a report that is designed to be read by a human being. It has nice section headers, and a formatted title.

Parameters **name** (*str*) – the title of the report

add_section(*heading*, *generator*, *index=None*)
Add a section to the report

This method adds a section with the given title, and generator to the report. An index may be specified to insert the section at a given location in the list; If no index is specified, the section is appended to the list. The view is called on the model which results from the generator when the report is run. A generator is simply a method or callable object which takes no arguments and returns a openstack.common.report.models.base.ReportModel or similar object.

The model is told to serialize as text (if possible) at serialization time by wrapping the generator. The view model's attached view (if any) is wrapped in a openstack.common.report.views.text.header.TitledView

Parameters

- **heading** (*str*) – the title for the section
- **generator** – the method or class which generates the model
- **index** (*int or None*) – the index at which to insert the section (or None to append)

3.7.550 The nova.openstack.common.report.utils Module

Various utilities for report generation

This module includes various utilities used in generating reports.

class StringWithAttrs
Bases: str

A String that can have arbitrary attributes

3.7.551 The nova.openstack.common.report.views.jinja_view Module

Provides Jinja Views

This module provides views that utilize the Jinja templating system for serialization. For more information on Jinja, please see http://jinja.pocoo.org/ .

class JinjaView(*path=None*, *text=None*)
Bases: object

A Jinja View

This view renders the given model using the provided Jinja template. The template can be given in various ways. If the *VIEw_TEXT* property is defined, that is used as template. Othewise, if a *path* parameter is passed to the constructor, that is used to load a file containing the template. If the *path* parameter is None, the *text* parameter is used as the template.

The leading newline character and trailing newline character are stripped from the template (provided they exist). Baseline indentation is also stripped from each line. The baseline indentation is determined by checking the indentation of the first line, after stripping off the leading newline (if any).

Parameters

- **path** (*str*) – the path to the Jinja template
- **text** (*str*) – the text of the Jinja template

template
Get the Compiled Template

Gets the compiled template, using a cached copy if possible (stored in attr:*_templatecache*) or otherwise recompiling the template if the compiled template is not present or is invalid (due to attr:*_regentemplate* being set to True).

Returns the compiled Jinja template

Return type jinja2.Template

text
Get the Template Text

Gets the text of the current template

Returns the text of the Jinja template

Return type str

3.7.552 The nova.openstack.common.report.views.json.generic Module

Provides generic JSON views

This modules defines several basic views for serializing data to JSON. Submodels that have already been serialized as JSON may have their string values marked with *__is_json__ = True* using `openstack.common.report.utils.StringWithAttrs` (each of the classes within this module does this automatically, and non-naive serializers check for this attribute and handle such strings specially)

class BasicKeyValueView

Bases: `object`

A Basic Key-Value JSON View

This view performs a naive serialization of a model into JSON by simply calling `json.dumps()` on the model

class KeyValueView

Bases: `object`

A Key-Value JSON View

This view performs advanced serialization to a model into JSON. It does so by first checking all values to see if they are marked as JSON. If so, they are deserialized using `json.loads()`. Then, the copy of the model with all JSON deserialized is reserialized into proper nested JSON using `json.dumps()`.

3.7.553 The nova.openstack.common.report.views.text.generic Module

Provides generic text views

This modules provides several generic views for serializing models into human-readable text.

class BasicKeyValueView

Bases: `object`

A Basic Key-Value Text View

This view performs a naive serialization of a model into text using a basic key-value method, where each key-value pair is rendered as "key = str(value)"

class KeyValueView(*indent_str=' '*, *key_sep=' = '*, *dict_sep=' = '*, *list_sep=' = '*, *anon_dict='[dict]'*, *before_dict=None*, *before_list=None*)

Bases: `object`

A Key-Value Text View

This view performs an advanced serialization of a model into text by following the following set of rules:

key [text] key = text

rootkey [Mapping]

```
rootkey =
  serialize(key, value)
```

key [Sequence]

```
key =
  serialize(item)
```

Parameters

- **indent_str** (*str*) – the string used to represent one "indent"

- **key_sep** (*str*) – the separator to use between keys and values
- **dict_sep** (*str*) – the separator to use after a dictionary root key
- **list_sep** (*str*) – the separator to use after a list root key
- **anon_dict** (*str*) – the "key" to use when there is a dict in a list (does not automatically use the dict separator)
- **before_dict** (*str or None*) – content to place on the line(s) before the a dict root key (use None to avoid inserting an extra line)
- **before_list** (*str or None*) – content to place on the line(s) before the a list root key (use None to avoid inserting an extra line)

class `MultiView`
Bases: `object`

A Text View Containing Multiple Views

This view simply serializes each value in the data model, and then joins them with newlines (ignoring the key values altogether). This is useful for serializing lists of models (as array-like dicts).

class `TableView`(*column_names*, *column_values*, *table_prop_name*)
Bases: `object`

A Basic Table Text View

This view performs serialization of data into a basic table with predefined column names and mappings. Column width is auto-calculated evenly, column values are automatically truncated accordingly. Values are centered in the columns.

Parameters

- **column_names** (*[str]*) – the headers for each of the columns
- **column_values** (*[str]*) – the item name to match each column to in each row
- **table_prop_name** (*str*) – the name of the property within the model containing the row models

3.7.554 The `nova.openstack.common.report.views.text.header` Module

Text Views With Headers

This package defines several text views with headers

class `HeaderView`(*header*)
Bases: `object`

A Text View With a Header

This view simply serializes the model and places the given header on top.

Parameters **header** – the header (can be anything on which str() can be called)

class `TitledView`(*title*)
Bases: `nova.openstack.common.report.views.text.header.HeaderView`

A Text View With a Title

This view simply serializes the model, and places a preformatted header containing the given title text on top. The title text can be up to 64 characters long.

Parameters **title** (*str*) – the title of the view

FORMAT_STR = '==========\n====

3.7.555 The nova.openstack.common.report.views.text.process Module

Provides process view

This module provides a view for visualizing processes in human-readable formm

class ProcessView (*path=None*, *text=None*)

Bases: nova.openstack.common.report.views.jinja_view.JinjaView

A Process View

This view displays process models defined by openstack.common.report.models.process.ProcessModel

VIEW_TEXT = "Process {{ pid }} (under {{ parent_pid }}) [run by: {{ username }} ({{ uids.real|default('unknown uid') }})

3.7.556 The nova.openstack.common.report.views.text.threading Module

Provides thread and stack-trace views

This module provides a collection of views for visualizing threads, green threads, and stack traces in human-readable form.

class GreenThreadView

Bases: object

A Green Thread View

This view displays a green thread provided by the data model openstack.common.report.models.threading.Green

FORMAT_STR = '——{thread_str: ^60}——\n{stack_trace}'

class StackTraceView (*path=None*, *text=None*)

Bases: nova.openstack.common.report.views.jinja_view.JinjaView

A Stack Trace View

This view displays stack trace models defined by openstack.common.report.models.threading.StackTraceMod

VIEW_TEXT = '{% if root_exception is not none %}Exception: {{ root_exception }}\n————————————————————\n\n{%

class ThreadView

Bases: object

A Thread Collection View

This view displays a python thread provided by the data model openstack.common.report.models.threading.ThreadModel # noqa

FORMAT_STR = '——{thread_str: ^60}——\n{stack_trace}'

3.7.557 The nova.openstack.common.report.views.xml.generic Module

Provides generic XML views

This modules defines several basic views for serializing data to XML. Submodels that have already been serialized as XML may have their string values marked with *__is_xml__ = True* using

`openstack.common.report.utils.StringWithAttrs` (each of the classes within this module does this automatically, and non-naive serializers check for this attribute and handle such strings specially)

class KeyValueView(*wrapper_name='model'*)

Bases: `object`

A Key-Value XML View

This view performs advanced serialization of a data model into XML. It first deserializes any values marked as XML so that they can be properly reserialized later. It then follows the following rules to perform serialization:

key [text/xml] The tag name is the key name, and the contents are the text or xml

key [Sequence] A wrapper tag is created with the key name, and each item is placed in an 'item' tag

key [Mapping] A wrapper tag is created with the key name, and the serialize is called on each key-value pair (such that each key gets its own tag)

Parameters **wrapper_name** (*str*) – the name of the top-level element

3.7.558 The `nova.openstack.common.service` Module

Generic Node base class for all workers that run on hosts.

class Launcher

Bases: `object`

Launch one or more services and wait for them to complete.

launch_service(*service*)

Load and start the given service.

Parameters **service** – The service you would like to start.

Returns None

restart()

Reload config files and restart service.

Returns None

stop()

Stop all services which are currently running.

Returns None

wait()

Waits until all services have been stopped, and then returns.

Returns None

class ProcessLauncher(*wait_interval=0.01*)

Bases: `object`

handle_signal()

launch_service(*service*, *workers=1*)

stop()

Terminate child processes and wait on each.

wait()

Loop waiting on children to die and respawning as necessary.

class Service (*threads=1000*)
Bases: object

Service object for binaries running on hosts.

reset ()

start ()

stop (*graceful=False*)

wait ()

class ServiceLauncher
Bases: nova.openstack.common.service.Launcher

handle_signal ()

wait (*ready_callback=None*)

class ServiceWrapper (*service*, *workers*)
Bases: object

class Services
Bases: object

add (*service*)

restart ()

static run_service (*service*, *done*)
Service start wrapper.

Parameters

- **service** – service to run
- **done** – event to wait on until a shutdown is triggered

Returns None

stop ()

wait ()

exception SignalExit (*signo*, *exccode=1*)
Bases: exceptions.SystemExit

launch (*service*, *workers=1*)

3.7.559 The nova.openstack.common.sslutils Module

is_enabled ()

list_opts ()
Entry point for oslo-config-generator.

wrap (*sock*)

3.7.560 The nova.openstack.common.systemd Module

Helper module for systemd service readiness notification.

notify()
Send notification to Systemd that service is ready.

For details see http://www.freedesktop.org/software/systemd/man/sd_notify.html

notify_once()
Send notification once to Systemd that service is ready.

Systemd sets NOTIFY_SOCKET environment variable with the name of the socket listening for notifications from services. This method removes the NOTIFY_SOCKET environment variable to ensure notification is sent only once.

onready(*notify_socket*, *timeout*)
Wait for systemd style notification on the socket.

Parameters

- **notify_socket** (*string*) – local socket address
- **timeout** (*float*) – socket timeout

Returns 0 service ready 1 service not ready 2 timeout occurred

3.7.561 The nova.openstack.common.threadgroup Module

class Thread(*thread*, *group*)
Bases: object

Wrapper around a greenthread, that holds a reference to the ThreadGroup. The Thread will notify the ThreadGroup when it has done so it can be removed from the threads list.

link(*func*, **args*, ***kwargs*)

stop()

wait()

class ThreadGroup(*thread_pool_size=10*)
Bases: object

The point of the ThreadGroup class is to:

•keep track of timers and greenthreads (making it easier to stop them when need be).

•provide an easy API to add timers.

add_dynamic_timer(*callback*, *initial_delay=None*, *periodic_interval_max=None*, **args*, ***kwargs*)

add_thread(*callback*, **args*, ***kwargs*)

add_timer(*interval*, *callback*, *initial_delay=None*, **args*, ***kwargs*)

stop(*graceful=False*)
stop function has the option of graceful=True/False.

•In case of graceful=True, wait for all threads to be finished. Never kill threads.

•In case of graceful=False, kill threads immediately.

stop_timers()

thread_done(*thread*)

wait()

3.7.562 The nova.opts Module

list_opts()

3.7.563 The nova.paths Module

basedir_def(**args*)
Return an uninterpolated path relative to $pybasedir.

basedir_rel(**args*)
Return a path relative to $pybasedir.

bindir_def(**args*)
Return an uninterpolated path relative to $bindir.

bindir_rel(**args*)
Return a path relative to $bindir.

state_path_def(**args*)
Return an uninterpolated path relative to $state_path.

state_path_rel(**args*)
Return a path relative to $state_path.

3.7.564 The nova.pci.device Module

allocate(*devobj*, *instance=None*)

check_device_status(*dev_status=None*)
Decorator to check device status before changing it.

claim(*devobj*, *instance=None*)

free(*devobj*, *instance=None*)

remove(*devobj*, *instance=None*)

3.7.565 The nova.pci.devspec Module

class PciAddress(*pci_addr*, *is_physical_function*)
Bases: object

Manages the address fields of the whitelist.

This class checks the address fields of the pci_passthrough_whitelist configuration option, validating the address fields. Example config are:

pci_passthrough_whitelist = {"address":"*:0a:00.*",
"physical_network":"physnet1"}
pci_passthrough_whitelist = {"vendor_id":"1137","product_id":"0071"}

This function class will validate the address fields, check for wildcards, and insert wildcards where the field is left blank.

match(*pci_addr*, *pci_phys_addr*)

class PciDeviceSpec(*dev_spec*)
Bases: object

get_tags()

match(*dev_dict*)

match_pci_obj(*pci_obj*)

get_pci_dev_info(*pci_obj*, *property*, *max*, *hex_value*)

get_value(*v*)

3.7.566 The nova.pci.manager Module

class PciDevTracker(*context*, *node_id=None*)
Bases: object

Manage pci devices in a compute node.

This class fetches pci passthrough information from hypervisor and trackes the usage of these devices.

It's called by compute node resource tracker to allocate and free devices to/from instances, and to update the available pci passthrough devices information from hypervisor periodically. The devices information is updated to DB when devices information is changed.

all_devs

clean_usage(*instances*, *migrations*, *orphans*)
Remove all usages for instances not passed in the parameter.

The caller should hold the COMPUTE_RESOURCE_SEMAPHORE lock

pci_stats

save(*context*)

set_hvdevs(*devices*)
Sync the pci device tracker with hypervisor information.

To support pci device hot plug, we sync with the hypervisor periodically, fetching all devices information from hypervisor, update the tracker and sync the DB information.

Devices should not be hot-plugged when assigned to a guest, but possibly the hypervisor has no such guarantee. The best we can do is to give a warning if a device is changed or removed while assigned.

update_pci_for_instance(*context*, *instance*)
Update instance's pci usage information.

The caller should hold the COMPUTE_RESOURCE_SEMAPHORE lock

update_pci_for_migration(*context*, *instance*, *sign=1*)
Update instance's pci usage information when it is migrated.

The caller should hold the COMPUTE_RESOURCE_SEMAPHORE lock.

Parameters **sign** – claim devices for instance when sign is 1, remove the claims when sign is -1

get_instance_pci_devs(*inst*, *request_id=None*)
Get the devices allocated to one or all requests for an instance.

- For generic PCI request, the request id is None.
- For sr-iov networking, the request id is a valid uuid
- There are a couple of cases where all the PCI devices allocated to an instance need to be returned. Refer to libvirt driver that handles soft_reboot and hard_boot of 'xen' instances.

3.7.567 The nova.pci.request Module

Example of a PCI alias:

```
| pci_alias = '{
|   "name": "QuicAssist",
|   "product_id": "0443",
|   "vendor_id": "8086",
|   "device_type": "ACCEL",
|   }'
```

Aliases with the same name and the same device_type are OR operation:

```
| pci_alias = '{
|   "name": "QuicAssist",
|   "product_id": "0442",
|   "vendor_id": "8086",
|   "device_type": "ACCEL",
|   }'
```

These 2 aliases define a device request meaning: vendor_id is "8086" and product id is "0442" or "0443".

get_pci_requests_from_flavor(*flavor*)

Get flavor's pci request.

The pci_passthrough:alias scope in flavor extra_specs describes the flavor's pci requests, the key is 'pci_passthrough:alias' and the value has format 'alias_name_x:count, alias_name_y:count, ... '. The alias_name is defined in 'pci_alias' configurations.

The flavor's requirement is translated into pci requests list, each entry in the list is a dictionary. The dictionary has three keys. The 'specs' gives the pci device properties requirement, the 'count' gives the number of devices, and the optional 'alias_name' is the corresponding alias definition name.

Example: Assume alias configuration is:

```
|   {'vendor_id':'8086',
|    'device_id':'1502',
|    'name':'alias_1'}
```

The flavor extra specs includes: 'pci_passthrough:alias': 'alias_1:2'.

The returned pci_requests are:

```
| pci_requests = [{'count':2,
|                 'specs': [{'vendor_id':'8086',
|                            'device_id':'1502'}],
|                 'alias_name': 'alias_1'}]
```

Parameters **flavor** – the flavor to be checked

Returns a list of pci requests

3.7.568 The nova.pci.stats Module

class PciDeviceStats(*stats=None*)

Bases: object

PCI devices summary information.

According to the PCI SR-IOV spec, a PCI physical function can have up to 256 PCI virtual functions, thus the number of assignable PCI functions in a cloud can be big. The scheduler needs to know all device availability information in order to determine which compute hosts can support a PCI request. Passing individual virtual device information to the scheduler does not scale, so we provide summary information.

Usually the virtual functions provided by a host PCI device have the same value for most properties, like vendor_id, product_id and class type. The PCI stats class summarizes this information for the scheduler.

The pci stats information is maintained exclusively by compute node resource tracker and updated to database. The scheduler fetches the information and selects the compute node accordingly. If a comptue node is selected, the resource tracker allocates the devices to the instance and updates the pci stats information.

This summary information will be helpful for cloud management also.

add_device(*dev*)
Add a device to its matching pool.

apply_requests(*requests*, *numa_cells=None*)
Apply PCI requests to the PCI stats.

This is used in multiple instance creation, when the scheduler has to maintain how the resources are consumed by the instances. If numa_cells is provided then only devices contained in those nodes are considered.

clear()
Clear all the stats maintained.

consume_requests(*pci_requests*, *numa_cells=None*)

get_free_devs()

static pool_cmp(*dev1*, *dev2*)

pool_keys = ['product_id', 'vendor_id', 'numa_node']

remove_device(*dev*)
Remove one device from the first pool that it matches.

support_requests(*requests*, *numa_cells=None*)
Check if the pci requests can be met.

Scheduler checks compute node's PCI stats to decide if an instance can be scheduled into the node. Support does not mean real allocation. If numa_cells is provided then only devices contained in those nodes are considered.

to_device_pools_obj()
Return the contents of the pools as a PciDevicePoolList object.

3.7.569 The nova.pci.utils Module

get_function_by_ifname(*ifname*)
Given the device name, returns the PCI address of a an device and returns True if the address in a physical function.

get_ifname_by_pci_address(*pci_addr*, *pf_interface=False*)
Get the interface name based on a VF's pci address

The returned interface name is either the parent PF's or that of the VF itself based on the argument of pf_interface.

get_pci_address_fields(*pci_addr*)

get_vf_num_by_pci_address(*pci_addr*)
Get the VF number based on a VF's pci address

A VF is associated with an VF number, which ip link command uses to configure it. This number can be obtained from the PCI device filesystem.

is_physical_function(*pci_addr*)

parse_address(*address*)
Returns (domain, bus, slot, function) from PCI address that is stored in PciDevice DB table.

pci_device_prop_match(*pci_dev*, *specs*)
Check if the pci_dev meet spec requirement

Specs is a list of PCI device property requirements. An example of device requirement that the PCI should be either: a) Device with vendor_id as 0x8086 and product_id as 0x8259, or b) Device with vendor_id as 0x10de and product_id as 0x10d8:

[{"vendor_id":"8086", "product_id":"8259"}, {"vendor_id":"10de", "product_id":"10d8"}]

3.7.570 The nova.pci.whitelist Module

class PciHostDevicesWhiteList(*whitelist_spec=None*)
Bases: object

White list class to decide assignable pci devices.

Not all devices on compute node can be assigned to guest, the cloud administrator decides the devices that can be assigned based on vendor_id or product_id etc. If no white list specified, no device will be assignable.

device_assignable(*dev*)
Check if a device can be assigned to a guest.

Parameters **dev** – A dictionary describing the device properties

get_devspec(*pci_dev*)

get_pci_device_devspec(*pci_dev*)

get_pci_devices_filter()

3.7.571 The nova.policy Module

Policy Engine For Nova.

class IsAdminCheck(*kind*, *match*)
Bases: nova.openstack.common.policy.Check

An explicit check for is_admin.

check_is_admin(*context*)
Whether or not roles contains 'admin' role according to policy setting.

enforce(*context*, *action*, *target*, *do_raise=True*, *exc=None*)
Verifies that the action is valid on the target in this context.

Parameters

- **context** – nova context

- **action** – string representing the action to be checked this should be colon separated for clarity. i.e. `compute:create_instance`, `compute:attach_volume`, `volume:attach_volume`
- **target** – dictionary representing the object of the action for object creation this should be a dictionary representing the location of the object e.g. `{'project_id': context.project_id}`
- **do_raise** – if True (the default), raises PolicyNotAuthorized; if False, returns False

Raises nova.exception.PolicyNotAuthorized if verification fails and do_raise is True.

Returns returns a non-False value (not necessarily "True") if authorized, and the exact value False if not authorized and do_raise is False.

get_rules()

init(*policy_file=None*, *rules=None*, *default_rule=None*, *use_conf=True*)
Init an Enforcer class.

Parameters

- **policy_file** – Custom policy file to use, if none is specified, *CONF.policy_file* will be used.
- **rules** – Default dictionary / Rules to use. It will be considered just in the first instantiation.
- **default_rule** – Default rule to use, CONF.default_rule will be used if none is specified.
- **use_conf** – Whether to load rules from config file.

reset()

set_rules(*rules*, *overwrite=True*, *use_conf=False*)
Set rules based on the provided dict of rules.

Parameters

- **rules** – New rules to use. It should be an instance of dict.
- **overwrite** – Whether to overwrite current rules or update them with the new rules.
- **use_conf** – Whether to reload rules from config file.

3.7.572 The nova.quota Module

Quotas for instances, and floating ips.

class AbsoluteResource(*name*, *flag=None*)
Bases: nova.quota.BaseResource

Describe a non-reservable resource.

valid_method = 'check'

class BaseResource(*name*, *flag=None*)
Bases: object

Describe a single resource for quota checking.

default
Return the default value of the quota.

quota(*driver*, *context*, ***kwargs*)
Given a driver and context, obtain the quota for this resource.

Parameters

- **driver** – A quota driver.
- **context** – The request context.
- **project_id** – The project to obtain the quota value for. If not provided, it is taken from the context. If it is given as None, no project-specific quota will be searched for.
- **quota_class** – The quota class corresponding to the project, or for which the quota is to be looked up. If not provided, it is taken from the context. If it is given as None, no quota class-specific quota will be searched for. Note that the quota class defaults to the value in the context, which may not correspond to the project if project_id is not the same as the one in the context.

class `CountableResource`(*name*, *count*, *flag=None*)
Bases: `nova.quota.AbsoluteResource`

Describe a resource where the counts aren't based solely on the project ID.

class `DbQuotaDriver`
Bases: `object`

Driver to perform necessary checks to enforce quotas and obtain quota information. The default driver utilizes the local database.

`UNLIMITED_VALUE = -1`

`commit`(*context*, *reservations*, *project_id=None*, *user_id=None*)
Commit reservations.

Parameters

- **context** – The request context, for access checks.
- **reservations** – A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

`destroy_all_by_project`(*context*, *project_id*)
Destroy all quotas, usages, and reservations associated with a project.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project being deleted.

`destroy_all_by_project_and_user`(*context*, *project_id*, *user_id*)
Destroy all quotas, usages, and reservations associated with a project and user.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project being deleted.
- **user_id** – The ID of the user being deleted.

`expire`(*context*)
Expire reservations.

Explores all currently existing reservations and rolls back any that have expired.

Parameters **context** – The request context, for access checks.

get_by_class(*context*, *quota_class*, *resource*)
Get a specific quota by quota class.

get_by_project(*context*, *project_id*, *resource*)
Get a specific quota by project.

get_by_project_and_user(*context*, *project_id*, *user_id*, *resource*)
Get a specific quota by project and user.

get_class_quotas(*context*, *resources*, *quota_class*, *defaults=True*)
Given a list of resources, retrieve the quotas for the given quota class.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **quota_class** – The name of the quota class to return quotas for.
- **defaults** – If True, the default value will be reported if there is no specific value for the resource.

get_defaults(*context*, *resources*)
Given a list of resources, retrieve the default quotas. Use the class quotas named *_DEFAULT_QUOTA_NAME* as default quotas, if it exists.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.

get_project_quotas(*context*, *resources*, *project_id*, *quota_class=None*, *defaults=True*, *usages=True*, *remains=False*, *project_quotas=None*)
Given a list of resources, retrieve the quotas for the given project.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **project_id** – The ID of the project to return quotas for.
- **quota_class** – If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified. It will be ignored if project_id == context.project_id.
- **defaults** – If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** – If True, the current in_use and reserved counts will also be returned.
- **remains** – If True, the current remains of the project will will be returned.
- **project_quotas** – Quotas dictionary for the specified project.

get_settable_quotas(*context*, *resources*, *project_id*, *user_id=None*)
Given a list of resources, retrieve the range of settable quotas for the given user or project.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **project_id** – The ID of the project to return quotas for.

- **user_id** – The ID of the user to return quotas for.

get_user_quotas(*context*, *resources*, *project_id*, *user_id*, *quota_class=None*, *defaults=True*, *usages=True*, *project_quotas=None*, *user_quotas=None*)

Given a list of resources, retrieve the quotas for the given user and project.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **project_id** – The ID of the project to return quotas for.
- **user_id** – The ID of the user to return quotas for.
- **quota_class** – If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified. It will be ignored if project_id == context.project_id.
- **defaults** – If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** – If True, the current in_use and reserved counts will also be returned.
- **project_quotas** – Quotas dictionary for the specified project.
- **user_quotas** – Quotas dictionary for the specified project and user.

limit_check(*context*, *resources*, *values*, *project_id=None*, *user_id=None*)

Check simple quota limits.

For limits–those quotas for which there is no usage synchronization function–this method checks that a set of proposed values are permitted by the limit restriction.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it is not a simple limit resource.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns nothing.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **values** – A dictionary of the values to check against the quota.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

reserve(*context*, *resources*, *deltas*, *expire=None*, *project_id=None*, *user_id=None*)

Check quotas and reserve resources.

For counting quotas–those quotas for which there is a usage synchronization function–this method checks quotas against current usage and the desired deltas.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it does not have a usage synchronization function.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns a list of reservation UUIDs which were created.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **deltas** – A dictionary of the proposed delta changes.
- **expire** – An optional parameter specifying an expiration time for the reservations. If it is a simple number, it is interpreted as a number of seconds and added to the current time; if it is a datetime.timedelta object, it will also be added to the current time. A datetime.datetime object will be interpreted as the absolute expiration time. If None is specified, the default expiration time set by –default-reservation-expire will be used (this value will be treated as a number of seconds).
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

rollback(*context*, *reservations*, *project_id=None*, *user_id=None*)

Roll back reservations.

Parameters

- **context** – The request context, for access checks.
- **reservations** – A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

usage_reset(*context*, *resources*)

Reset the usage records for a particular user on a list of resources. This will force that user's usage records to be refreshed the next time a reservation is made.

Note: this does not affect the currently outstanding reservations the user has; those reservations must be committed or rolled back (or expired).

Parameters

- **context** – The request context, for access checks.
- **resources** – A list of the resource names for which the usage must be reset.

class NoopQuotaDriver

Bases: `object`

Driver that turns quotas calls into no-ops and pretends that quotas for all resources are unlimited. This can be used if you do not wish to have any quota checking. For instance, with nova compute cells, the parent cell should do quota checking, but the child cell should not.

commit(*context*, *reservations*, *project_id=None*, *user_id=None*)

Commit reservations.

Parameters

- **context** – The request context, for access checks.
- **reservations** – A list of the reservation UUIDs, as returned by the reserve() method.

- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

destroy_all_by_project(*context*, *project_id*)
Destroy all quotas, usages, and reservations associated with a project.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project being deleted.

destroy_all_by_project_and_user(*context*, *project_id*, *user_id*)
Destroy all quotas, usages, and reservations associated with a project and user.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project being deleted.
- **user_id** – The ID of the user being deleted.

expire(*context*)
Expire reservations.

Explores all currently existing reservations and rolls back any that have expired.

Parameters **context** – The request context, for access checks.

get_by_class(*context*, *quota_class*, *resource*)
Get a specific quota by quota class.

get_by_project(*context*, *project_id*, *resource*)
Get a specific quota by project.

get_by_project_and_user(*context*, *project_id*, *user_id*, *resource*)
Get a specific quota by project and user.

get_class_quotas(*context*, *resources*, *quota_class*, *defaults=True*)
Given a list of resources, retrieve the quotas for the given quota class.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **quota_class** – The name of the quota class to return quotas for.
- **defaults** – If True, the default value will be reported if there is no specific value for the resource.

get_defaults(*context*, *resources*)
Given a list of resources, retrieve the default quotas.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.

get_project_quotas(*context*, *resources*, *project_id*, *quota_class=None*, *defaults=True*, *usages=True*, *remains=False*)
Given a list of resources, retrieve the quotas for the given project.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **project_id** – The ID of the project to return quotas for.
- **quota_class** – If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified. It will be ignored if project_id == context.project_id.
- **defaults** – If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** – If True, the current in_use and reserved counts will also be returned.
- **remains** – If True, the current remains of the project will will be returned.

get_settable_quotas(*context*, *resources*, *project_id*, *user_id=None*)
Given a list of resources, retrieve the range of settable quotas for the given user or project.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **project_id** – The ID of the project to return quotas for.
- **user_id** – The ID of the user to return quotas for.

get_user_quotas(*context*, *resources*, *project_id*, *user_id*, *quota_class=None*, *defaults=True*, *usages=True*)
Given a list of resources, retrieve the quotas for the given user and project.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **project_id** – The ID of the project to return quotas for.
- **user_id** – The ID of the user to return quotas for.
- **quota_class** – If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified. It will be ignored if project_id == context.project_id.
- **defaults** – If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** – If True, the current in_use and reserved counts will also be returned.

limit_check(*context*, *resources*, *values*, *project_id=None*, *user_id=None*)
Check simple quota limits.

For limits–those quotas for which there is no usage synchronization function–this method checks that a set of proposed values are permitted by the limit restriction.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it is not a simple limit resource.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns nothing.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **values** – A dictionary of the values to check against the quota.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

reserve(*context*, *resources*, *deltas*, *expire=None*, *project_id=None*, *user_id=None*)
Check quotas and reserve resources.

For counting quotas–those quotas for which there is a usage synchronization function–this method checks quotas against current usage and the desired deltas.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it does not have a usage synchronization function.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns a list of reservation UUIDs which were created.

Parameters

- **context** – The request context, for access checks.
- **resources** – A dictionary of the registered resources.
- **deltas** – A dictionary of the proposed delta changes.
- **expire** – An optional parameter specifying an expiration time for the reservations. If it is a simple number, it is interpreted as a number of seconds and added to the current time; if it is a datetime.timedelta object, it will also be added to the current time. A datetime.datetime object will be interpreted as the absolute expiration time. If None is specified, the default expiration time set by –default-reservation-expire will be used (this value will be treated as a number of seconds).
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

rollback(*context*, *reservations*, *project_id=None*, *user_id=None*)
Roll back reservations.

Parameters

- **context** – The request context, for access checks.
- **reservations** – A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

usage_reset(*context*, *resources*)
Reset the usage records for a particular user on a list of resources. This will force that user's usage records to be refreshed the next time a reservation is made.

Note: this does not affect the currently outstanding reservations the user has; those reservations must be committed or rolled back (or expired).

Parameters

- **context** – The request context, for access checks.
- **resources** – A list of the resource names for which the usage must be reset.

class `QuotaEngine`(*quota_driver_class=None*)

Bases: `object`

Represent the set of recognized quotas.

`commit`(*context*, *reservations*, *project_id=None*, *user_id=None*)

Commit reservations.

Parameters

- **context** – The request context, for access checks.
- **reservations** – A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.

`count`(*context*, *resource*, **args*, ***kwargs*)

Count a resource.

For countable resources, invokes the count() function and returns its result. Arguments following the context and resource are passed directly to the count function declared by the resource.

Parameters

- **context** – The request context, for access checks.
- **resource** – The name of the resource, as a string.

`destroy_all_by_project`(*context*, *project_id*)

Destroy all quotas, usages, and reservations associated with a project.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project being deleted.

`destroy_all_by_project_and_user`(*context*, *project_id*, *user_id*)

Destroy all quotas, usages, and reservations associated with a project and user.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project being deleted.
- **user_id** – The ID of the user being deleted.

`expire`(*context*)

Expire reservations.

Explores all currently existing reservations and rolls back any that have expired.

Parameters **context** – The request context, for access checks.

`get_by_class`(*context*, *quota_class*, *resource*)

Get a specific quota by quota class.

get_by_project(*context*, *project_id*, *resource*)
Get a specific quota by project.

get_by_project_and_user(*context*, *project_id*, *user_id*, *resource*)
Get a specific quota by project and user.

get_class_quotas(*context*, *quota_class*, *defaults=True*)
Retrieve the quotas for the given quota class.

Parameters

- **context** – The request context, for access checks.
- **quota_class** – The name of the quota class to return quotas for.
- **defaults** – If True, the default value will be reported if there is no specific value for the resource.

get_defaults(*context*)
Retrieve the default quotas.

Parameters **context** – The request context, for access checks.

get_project_quotas(*context*, *project_id*, *quota_class=None*, *defaults=True*, *usages=True*, *remains=False*)
Retrieve the quotas for the given project.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project to return quotas for.
- **quota_class** – If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified.
- **defaults** – If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** – If True, the current in_use and reserved counts will also be returned.
- **remains** – If True, the current remains of the project will will be returned.

get_settable_quotas(*context*, *project_id*, *user_id=None*)
Given a list of resources, retrieve the range of settable quotas for the given user or project.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project to return quotas for.
- **user_id** – The ID of the user to return quotas for.

get_user_quotas(*context*, *project_id*, *user_id*, *quota_class=None*, *defaults=True*, *usages=True*)
Retrieve the quotas for the given user and project.

Parameters

- **context** – The request context, for access checks.
- **project_id** – The ID of the project to return quotas for.
- **user_id** – The ID of the user to return quotas for.
- **quota_class** – If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified.

- **defaults** – If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** – If True, the current in_use and reserved counts will also be returned.

limit_check(*context*, *project_id=None*, *user_id=None*, ***values*)
Check simple quota limits.

For limits–those quotas for which there is no usage synchronization function–this method checks that a set of proposed values are permitted by the limit restriction. The values to check are given as keyword arguments, where the key identifies the specific quota limit to check, and the value is the proposed value.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it is not a simple limit resource.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns nothing.

Parameters

- **context** – The request context, for access checks.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.
- **user_id** – Specify the user_id if current context is admin and admin wants to impact on common user.

register_resource(*resource*)
Register a resource.

register_resources(*resources*)
Register a list of resources.

reserve(*context*, *expire=None*, *project_id=None*, *user_id=None*, ***deltas*)
Check quotas and reserve resources.

For counting quotas–those quotas for which there is a usage synchronization function–this method checks quotas against current usage and the desired deltas. The deltas are given as keyword arguments, and current usage and other reservations are factored into the quota check.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it does not have a usage synchronization function.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns a list of reservation UUIDs which were created.

Parameters

- **context** – The request context, for access checks.
- **expire** – An optional parameter specifying an expiration time for the reservations. If it is a simple number, it is interpreted as a number of seconds and added to the current time; if it is a datetime.timedelta object, it will also be added to the current time. A datetime.datetime object will be interpreted as the absolute expiration time. If None is specified, the default expiration time set by –default-reservation-expire will be used (this value will be treated as a number of seconds).
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.

resources

rollback(*context*, *reservations*, *project_id=None*, *user_id=None*)
Roll back reservations.

Parameters

- **context** – The request context, for access checks.
- **reservations** – A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** – Specify the project_id if current context is admin and admin wants to impact on common user's tenant.

usage_reset(*context*, *resources*)
Reset the usage records for a particular user on a list of resources. This will force that user's usage records to be refreshed the next time a reservation is made.

Note: this does not affect the currently outstanding reservations the user has; those reservations must be committed or rolled back (or expired).

Parameters

- **context** – The request context, for access checks.
- **resources** – A list of the resource names for which the usage must be reset.

class **ReservableResource**(*name*, *sync*, *flag=None*)
Bases: nova.quota.BaseResource

Describe a reservable resource.

valid_method = 'reserve'

3.7.573 The nova.rpc Module

init(*conf*)

cleanup()

set_defaults(*control_exchange*)

add_extra_exmods(**args*)

clear_extra_exmods()

get_allowed_exmods()

class **RequestContextSerializer**(*base*)
Bases: oslo_messaging.serializer.Serializer

deserialize_context(*context*)

deserialize_entity(*context*, *entity*)

serialize_context(*context*)

serialize_entity(*context*, *entity*)

get_client(*target*, *version_cap=None*, *serializer=None*)

get_server(*target*, *endpoints*, *serializer=None*)

get_notifier(*service*, *host=None*, *publisher_id=None*)

3.7.574 The nova.safe_utils Module

Utilities and helper functions that won't produce circular imports.

getcallargs (*function*, **args*, ***kwargs*)
This is a simplified inspect.getcallargs (2.7+).

It should be replaced when python >= 2.7 is standard.

This method can only properly grab arguments which are passed in as keyword arguments, or given names by the method being called. This means that an `*arg` in a method signature and any arguments captured by it will be left out of the results.

3.7.575 The nova.scheduler.caching_scheduler Module

class CachingScheduler (**args*, ***kwargs*)
Bases: nova.scheduler.filter_scheduler.FilterScheduler

Scheduler to test aggressive caching of the host list.

Please note, this is a very opinionated scheduler. Be sure to review the caveats listed here before selecting this scheduler.

The aim of this scheduler is to reduce server build times when you have large bursts of server builds, by reducing the time it takes, from the users point of view, to service each schedule request.

There are two main parts to scheduling a users request: * getting the current state of the system * using filters and weights to pick the best host

This scheduler tries its best to cache in memory the current state of the system, so we don't need to make the expensive call to get the current state of the system while processing a user's request, we can do that query in a periodic task before the user even issues their request.

To reduce races, cached info of the chosen host is updated using the existing host state call: consume_from_instance

Please note, the way this works, each scheduler worker has its own copy of the cache. So if you run multiple schedulers, you will get more retries, because the data stored on any additional scheduler will be more out of date, than if it was fetched from the database.

In a similar way, if you have a high number of server deletes, the extra capacity from those deletes will not show up until the cache is refreshed.

run_periodic_tasks (*context*)
Called from a periodic tasks in the manager.

3.7.576 The nova.scheduler.chance Module

Chance (Random) Scheduler implementation

class ChanceScheduler
Bases: nova.scheduler.driver.Scheduler

Implements Scheduler as a random node selector.

select_destinations (*context*, *request_spec*, *filter_properties*)
Selects random destinations.

3.7.577 The nova.scheduler.client.query Module

class SchedulerQueryClient
Bases: object

Client class for querying to the scheduler.

delete_aggregate(*context*, *aggregate*)
Deletes HostManager internal information about a specific aggregate.

Parameters **aggregate** (nova.objects.Aggregate) – Aggregate to delete

delete_instance_info(*context*, *host_name*, *instance_uuid*)
Updates the HostManager with the current information about an instance that has been deleted on a host.

Parameters

- **context** – local context
- **host_name** – name of host sending the update
- **instance_uuid** – the uuid of the deleted instance

select_destinations(*context*, *request_spec*, *filter_properties*)
Returns destinations(s) best suited for this request_spec and filter_properties.

The result should be a list of dicts with 'host', 'nodename' and 'limits' as keys.

sync_instance_info(*context*, *host_name*, *instance_uuids*)
Notifies the HostManager of the current instances on a host by sending a list of the uuids for those instances. The HostManager can then compare that with its in-memory view of the instances to detect when they are out of sync.

Parameters

- **context** – local context
- **host_name** – name of host sending the update
- **instance_uuids** – a list of UUID strings representing the current instances on the specified host

update_aggregates(*context*, *aggregates*)
Updates HostManager internal aggregates information.

Parameters **aggregates** (nova.objects.Aggregate or nova.objects.AggregateList) – Aggregate(s) to update

update_instance_info(*context*, *host_name*, *instance_info*)
Updates the HostManager with the current information about the instances on a host.

Parameters

- **context** – local context
- **host_name** – name of host sending the update
- **instance_info** – an InstanceList object.

3.7.578 The nova.scheduler.client.report Module

class SchedulerReportClient
Bases: object

Client class for updating the scheduler.

update_resource_stats(*compute_node*)
Creates or updates stats for the supplied compute node.

Parameters **compute_node** – updated nova.objects.ComputeNode to report

3.7.579 The nova.scheduler.driver Module

Scheduler base class that all Schedulers should inherit from

class Scheduler
Bases: object

The base class that all Scheduler classes should inherit from.

hosts_up(*context*, *topic*)
Return the list of hosts that have a running service for topic.

run_periodic_tasks(*context*)
Manager calls this so drivers can perform periodic tasks.

select_destinations(*context*, *request_spec*, *filter_properties*)
Must override select_destinations method.

Returns A list of dicts with 'host', 'nodename' and 'limits' as keys that satisfies the request_spec and filter_properties.

3.7.580 The nova.scheduler.filter_scheduler Module

The FilterScheduler is for creating instances locally. You can customize this scheduler by specifying your own Host Filters and Weighing Functions.

class FilterScheduler(**args*, ***kwargs*)
Bases: nova.scheduler.driver.Scheduler

Scheduler that can be used for filtering and weighing.

populate_filter_properties(*request_spec*, *filter_properties*)
Stuff things into filter_properties. Can be overridden in a subclass to add more data.

select_destinations(*context*, *request_spec*, *filter_properties*)
Selects a filtered set of hosts and nodes.

3.7.581 The nova.scheduler.filters.affinity_filter Module

class DifferentHostFilter
Bases: nova.scheduler.filters.BaseHostFilter

Schedule the instance on a different host from a set of instances.

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

class SameHostFilter
Bases: nova.scheduler.filters.BaseHostFilter

Schedule the instance on the same host as another instance in a set of instances.

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

class ServerGroupAffinityFilter
Bases: nova.scheduler.filters.affinity_filter._GroupAffinityFilter

class ServerGroupAntiAffinityFilter
Bases: nova.scheduler.filters.affinity_filter._GroupAntiAffinityFilter

class SimpleCIDRAffinityFilter
Bases: nova.scheduler.filters.BaseHostFilter

Schedule the instance on a host with a particular cidr

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

3.7.582 The nova.scheduler.filters.aggregate_image_properties_isolation Module

class AggregateImagePropertiesIsolation
Bases: nova.scheduler.filters.BaseHostFilter

AggregateImagePropertiesIsolation works with image properties.

host_passes(*host_state*, *filter_properties*)
Checks a host in an aggregate that metadata key/value match with image properties.

run_filter_once_per_request = True

3.7.583 The nova.scheduler.filters.aggregate_instance_extra_specs Module

class AggregateInstanceExtraSpecsFilter
Bases: nova.scheduler.filters.BaseHostFilter

AggregateInstanceExtraSpecsFilter works with InstanceType records.

host_passes(*host_state*, *filter_properties*)
Return a list of hosts that can create instance_type

Check that the extra specs associated with the instance type match the metadata provided by aggregates. If not present return False.

run_filter_once_per_request = True

3.7.584 The nova.scheduler.filters.aggregate_multitenancy_isolation Module

class AggregateMultiTenancyIsolation
Bases: nova.scheduler.filters.BaseHostFilter

Isolate tenants in specific aggregates.

host_passes(*host_state*, *filter_properties*)
If a host is in an aggregate that has the metadata key "filter_tenant_id" it can only create instances from that tenant(s). A host can be in different aggregates.

If a host doesn't belong to an aggregate with the metadata key "filter_tenant_id" it can create instances from all tenants.

run_filter_once_per_request = True

3.7.585 The nova.scheduler.filters.all_hosts_filter Module

class AllHostsFilter

Bases: nova.scheduler.filters.BaseHostFilter

NOOP host filter. Returns all hosts.

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

3.7.586 The nova.scheduler.filters.availability_zone_filter Module

class AvailabilityZoneFilter

Bases: nova.scheduler.filters.BaseHostFilter

Filters Hosts by availability zone.

Works with aggregate metadata availability zones, using the key 'availability_zone' Note: in theory a compute node can be part of multiple availability_zones

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

3.7.587 The nova.scheduler.filters.compute_capabilities_filter Module

class ComputeCapabilitiesFilter

Bases: nova.scheduler.filters.BaseHostFilter

HostFilter hard-coded to work with InstanceType records.

host_passes(*host_state*, *filter_properties*)

Return a list of hosts that can create instance_type.

run_filter_once_per_request = True

3.7.588 The nova.scheduler.filters.compute_filter Module

class ComputeFilter

Bases: nova.scheduler.filters.BaseHostFilter

Filter on active Compute nodes.

host_passes(*host_state*, *filter_properties*)

Returns True for only active compute nodes.

run_filter_once_per_request = True

3.7.589 The nova.scheduler.filters.core_filter Module

class AggregateCoreFilter
Bases: nova.scheduler.filters.core_filter.BaseCoreFilter

AggregateCoreFilter with per-aggregate CPU subscription flag.

Fall back to global cpu_allocation_ratio if no per-aggregate setting found.

class BaseCoreFilter
Bases: nova.scheduler.filters.BaseHostFilter

host_passes(*host_state*, *filter_properties*)
Return True if host has sufficient CPU cores.

class CoreFilter
Bases: nova.scheduler.filters.core_filter.BaseCoreFilter

CoreFilter filters based on CPU core utilization.

3.7.590 The nova.scheduler.filters.disk_filter Module

class AggregateDiskFilter
Bases: nova.scheduler.filters.disk_filter.DiskFilter

AggregateDiskFilter with per-aggregate disk allocation ratio flag.

Fall back to global disk_allocation_ratio if no per-aggregate setting found.

class DiskFilter
Bases: nova.scheduler.filters.BaseHostFilter

Disk Filter with over subscription flag.

host_passes(*host_state*, *filter_properties*)
Filter based on disk usage.

3.7.591 The nova.scheduler.filters.exact_core_filter Module

class ExactCoreFilter
Bases: nova.scheduler.filters.BaseHostFilter

Exact Core Filter.

host_passes(*host_state*, *filter_properties*)
Return True if host has the exact number of CPU cores.

3.7.592 The nova.scheduler.filters.exact_disk_filter Module

class ExactDiskFilter
Bases: nova.scheduler.filters.BaseHostFilter

Exact Disk Filter.

host_passes(*host_state*, *filter_properties*)
Return True if host has the exact amount of disk available.

3.7.593 The nova.scheduler.filters.exact_ram_filter Module

class ExactRamFilter
Bases: nova.scheduler.filters.BaseHostFilter

Exact RAM Filter.

host_passes (*host_state*, *filter_properties*)
Return True if host has the exact amount of RAM available.

3.7.594 The nova.scheduler.filters.extra_specs_ops Module

match (*value*, *req*)

3.7.595 The nova.scheduler.filters.image_props_filter Module

class ImagePropertiesFilter
Bases: nova.scheduler.filters.BaseHostFilter

Filter compute nodes that satisfy instance image properties.

The ImagePropertiesFilter filters compute nodes that satisfy any architecture, hypervisor type, or virtual machine mode properties specified on the instance's image properties. Image properties are contained in the image dictionary in the request_spec.

host_passes (*host_state*, *filter_properties*)
Check if host passes specified image properties.

Returns True for compute nodes that satisfy image properties contained in the request_spec.

run_filter_once_per_request = True

3.7.596 The nova.scheduler.filters.io_ops_filter Module

class AggregateIoOpsFilter
Bases: nova.scheduler.filters.io_ops_filter.IoOpsFilter

AggregateIoOpsFilter with per-aggregate the max io operations.

Fall back to global max_io_ops_per_host if no per-aggregate setting found.

class IoOpsFilter
Bases: nova.scheduler.filters.BaseHostFilter

Filter out hosts with too many concurrent I/O operations.

host_passes (*host_state*, *filter_properties*)
Use information about current vm and task states collected from compute node statistics to decide whether to filter.

3.7.597 The nova.scheduler.filters.isolated_hosts_filter Module

class IsolatedHostsFilter
Bases: nova.scheduler.filters.BaseHostFilter

Keep specified images to selected hosts.

host_passes(*host_state*, *filter_properties*)
Result Matrix with 'restrict_isolated_hosts_to_isolated_images' set to True:

```
|                | isolated_image | non_isolated_image
|   -------------+----------------+--------------------
|   iso_host     |    True        |     False
|   non_iso_host |    False       |      True
```

Result Matrix with 'restrict_isolated_hosts_to_isolated_images' set to False:

```
|                | isolated_image | non_isolated_image
|   -------------+----------------+--------------------
|   iso_host     |    True        |      True
|   non_iso_host |    False       |      True
```

run_filter_once_per_request = True

3.7.598 The nova.scheduler.filters.json_filter Module

class JsonFilter
Bases: nova.scheduler.filters.BaseHostFilter

Host Filter to allow simple JSON-based grammar for selecting hosts.

commands = {'and': <function _and at 0x7f8577344cf8>, 'or': <function _or at 0x7f8577344c80>, '>=': <function _greate

host_passes(*host_state*, *filter_properties*)
Return a list of hosts that can fulfill the requirements specified in the query.

3.7.599 The nova.scheduler.filters.metrics_filter Module

class MetricsFilter
Bases: nova.scheduler.filters.BaseHostFilter

Metrics Filter

This filter is used to filter out those hosts which don't have the corresponding metrics so these the metrics weigher won't fail due to these hosts.

host_passes(*host_state*, *filter_properties*)

3.7.600 The nova.scheduler.filters.num_instances_filter Module

class AggregateNumInstancesFilter
Bases: nova.scheduler.filters.num_instances_filter.NumInstancesFilter

AggregateNumInstancesFilter with per-aggregate the max num instances.

Fall back to global max_num_instances_per_host if no per-aggregate setting found.

class NumInstancesFilter
Bases: nova.scheduler.filters.BaseHostFilter

Filter out hosts with too many instances.

host_passes(*host_state*, *filter_properties*)

3.7.601 The nova.scheduler.filters.numa_topology_filter Module

class NUMATopologyFilter
Bases: nova.scheduler.filters.BaseHostFilter

Filter on requested NUMA topology.

host_passes(*host_state*, *filter_properties*)

3.7.602 The nova.scheduler.filters.pci_passthrough_filter Module

class PciPassthroughFilter
Bases: nova.scheduler.filters.BaseHostFilter

Pci Passthrough Filter based on PCI request

Filter that schedules instances on a host if the host has devices to meet the device requests in the 'extra_specs' for the flavor.

PCI resource tracker provides updated summary information about the PCI devices for each host, like:

```
| [{"count": 5, "vendor_id": "8086", "product_id": "1520",
|   "extra_info":'{}'}],
```

and VM requests PCI devices via PCI requests, like:

```
| [{"count": 1, "vendor_id": "8086", "product_id": "1520",}].
```

The filter checks if the host passes or not based on this information.

host_passes(*host_state*, *filter_properties*)
Return true if the host has the required PCI devices.

3.7.603 The nova.scheduler.filters.ram_filter Module

class AggregateRamFilter
Bases: nova.scheduler.filters.ram_filter.BaseRamFilter

AggregateRamFilter with per-aggregate ram subscription flag.

Fall back to global ram_allocation_ratio if no per-aggregate setting found.

class BaseRamFilter
Bases: nova.scheduler.filters.BaseHostFilter

host_passes(*host_state*, *filter_properties*)
Only return hosts with sufficient available RAM.

class RamFilter
Bases: nova.scheduler.filters.ram_filter.BaseRamFilter

Ram Filter with over subscription flag.

3.7.604 The nova.scheduler.filters.retry_filter Module

class RetryFilter
Bases: nova.scheduler.filters.BaseHostFilter

Filter out nodes that have already been attempted for scheduling purposes

host_passes(*host_state*, *filter_properties*)
Skip nodes that have already been attempted.

3.7.605 The nova.scheduler.filters.trusted_filter Module

Filter to add support for Trusted Computing Pools.

Filter that only schedules tasks on a host if the integrity (trust) of that host matches the trust requested in the `extra_specs` for the flavor. The `extra_specs` will contain a key/value pair where the key is `trust`. The value of this pair (`trusted`/`untrusted`) must match the integrity of that host (obtained from the Attestation service) before the task can be scheduled on that host.

Note that the parameters to control access to the Attestation Service are in the `nova.conf` file in a separate `trust` section. For example, the config file will look something like:

[DEFAULT] verbose=True ... [trust] server=attester.mynetwork.com

Details on the specific parameters can be found in the file `trust_attest.py`.

Details on setting up and using an Attestation Service can be found at the Open Attestation project at:

https://github.com/OpenAttestation/OpenAttestation

class AttestationService
Bases: `object`

do_attestation(*hosts*)
Attests compute nodes through OAT service.

Parameters **hosts** – hosts list to be attested

Returns dictionary for trust level and validate time

class ComputeAttestation
Bases: `object`

is_trusted(*host*, *trust*)

class ComputeAttestationCache
Bases: `object`

Cache for compute node attestation

Cache compute node's trust level for sometime, if the cache is out of date, poll OAT service to flush the cache.

OAT service may have cache also. OAT service's cache valid time should be set shorter than trusted filter's cache valid time.

get_host_attestation(*host*)
Check host's trust level.

class TrustedFilter
Bases: `nova.scheduler.filters.BaseHostFilter`

Trusted filter to support Trusted Compute Pools.

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

3.7.606 The nova.scheduler.filters.type_filter Module

class AggregateTypeAffinityFilter
Bases: nova.scheduler.filters.BaseHostFilter

AggregateTypeAffinityFilter limits instance_type by aggregate

return True if no instance_type key is set or if the aggregate metadata key 'instance_type' has the instance_type name as a value

host_passes(*host_state*, *filter_properties*)

run_filter_once_per_request = True

class TypeAffinityFilter
Bases: nova.scheduler.filters.BaseHostFilter

TypeAffinityFilter doesn't allow more than one VM type per host.

Note: this works best with ram_weight_multiplier (spread) set to 1 (default).

host_passes(*host_state*, *filter_properties*)
Dynamically limits hosts to one instance type

Return False if host has any instance types other than the requested type. Return True if all instance types match or if host is empty.

3.7.607 The nova.scheduler.filters.utils Module

Bench of utility methods used by filters.

aggregate_metadata_get_by_host(*host_state*, *key=None*)
Returns a dict of all metadata based on a metadata key for a specific host. If the key is not provided, returns a dict of all metadata.

aggregate_values_from_key(*host_state*, *key_name*)
Returns a set of values based on a metadata key for a specific host.

instance_uuids_overlap(*host_state*, *uuids*)
Tests for overlap between a host_state and a list of uuids.

Returns True if any of the supplied uuids match any of the instance.uuid values in the host_state.

other_types_on_host(*host_state*, *instance_type_id*)
Tests for overlap between a host_state's instances and an instance_type_id.

Returns True if there are any instances in the host_state whose instance_type_id is different than the supplied instance_type_id value.

validate_num_values(*vals*, *default=None*, *cast_to=<type 'int'>*, *based_on=<built-in function min>*)
Returns a correctly casted value based on a set of values.

This method is useful to work with per-aggregate filters, It takes a set of values then return the 'based_on'{min/max} converted to 'cast_to' of the set or the default value.

Note: The cast implies a possible ValueError

3.7.608 The nova.scheduler.host_manager Module

Manage hosts in the current zone.

class HostManager
Bases: object

Base HostManager class.

delete_aggregate(*aggregate*)
Deletes internal HostManager information about a specific aggregate.

delete_instance_info(**args*, ***kwargs*)
Receives the UUID from a compute node when one of its instances is terminated.

The instance in the local view of the host's instances is removed.

get_all_host_states(*context*)
Returns a list of HostStates that represents all the hosts the HostManager knows about. Also, each of the consumable resources in HostState are pre-populated and adjusted based on data in the db.

get_filtered_hosts(*hosts*, *filter_properties*, *filter_class_names=None*, *index=0*)
Filter hosts and return only ones passing all filters.

get_weighed_hosts(*hosts*, *weight_properties*)
Weigh the hosts.

host_state_cls(*host*, *node*, ***kwargs*)

sync_instance_info(**args*, ***kwargs*)
Receives the uuids of the instances on a host.

This method is periodically called by the compute nodes, which send a list of all the UUID values for the instances on that node. This is used by the scheduler's HostManager to detect when its view of the compute node's instances is out of sync.

update_aggregates(*aggregates*)
Updates internal HostManager information about aggregates.

update_instance_info(**args*, ***kwargs*)
Receives an InstanceList object from a compute node.

This method receives information from a compute node when it starts up, or when its instances have changed, and updates its view of hosts and instances with it.

class HostState(*host*, *node*, *compute=None*)
Bases: object

Mutable and immutable information tracked for a host. This is an attempt to remove the ad-hoc data structures previously used and lock down access.

consume_from_instance(*instance*)
Incrementally update host state from an instance.

update_from_compute_node(*compute*)
Update information about a host from a ComputeNode object.

update_service(*service*)

class MetricItem
Bases: tuple

MetricItem(value, timestamp, source)

source
Alias for field number 2

timestamp
Alias for field number 1

value
Alias for field number 0

class ReadOnlyDict(*source=None*)
Bases: UserDict.IterableUserDict

A read-only dict.

clear()

pop(*key*, **args*)

popitem()

update()

3.7.609 The nova.scheduler.ironic_host_manager Module

Ironic host manager.

This host manager will consume all cpu's, disk space, and ram from a host / node as it is supporting Baremetal hosts, which can not be subdivided into multiple instances.

class IronicHostManager
Bases: nova.scheduler.host_manager.HostManager

Ironic HostManager class.

host_state_cls(*host*, *node*, ***kwargs*)
Factory function/property to create a new HostState.

class IronicNodeState(*host*, *node*, *compute=None*)
Bases: nova.scheduler.host_manager.HostState

Mutable and immutable information tracked for a host. This is an attempt to remove the ad-hoc data structures previously used and lock down access.

consume_from_instance(*instance*)
Consume nodes entire resources regardless of instance request.

update_from_compute_node(*compute*)
Update information about a host from a ComputeNode object.

3.7.610 The nova.scheduler.manager Module

Scheduler Service

class SchedulerManager(*scheduler_driver=None*, **args*, ***kwargs*)
Bases: nova.manager.Manager

Chooses a host to run instances on.

delete_aggregate(*ctxt*, *aggregate*)
Deletes HostManager internal information about a specific aggregate.

Parameters **aggregate** (nova.objects.Aggregate) – Aggregate to delete

delete_instance_info(*context*, *host_name*, *instance_uuid*)
Receives information about the deletion of one of a host's instances, and updates the driver's HostManager with that information.

select_destinations(**args*, ***kwargs*)

sync_instance_info(*context*, *host_name*, *instance_uuids*)
Receives a sync request from a host, and passes it on to the driver's HostManager.

target = <Target version=4.2>

update_aggregates(*ctxt*, *aggregates*)
Updates HostManager internal aggregates information.

Parameters **aggregates** (`nova.objects.Aggregate` or `nova.objects.AggregateList`) – Aggregate(s) to update

update_instance_info(*context*, *host_name*, *instance_info*)
Receives information about changes to a host's instances, and updates the driver's HostManager with that information.

3.7.611 The `nova.scheduler.opts` Module

list_opts()

3.7.612 The `nova.scheduler.rpcapi` Module

Client side of the scheduler manager RPC API.

class SchedulerAPI
Bases: `object`

Client side of the scheduler rpc API.

API version history:

- 1.0 - Initial version.
- **1.1 - Changes to prep_resize():**
  - remove instance_uuid, add instance
  - remove instance_type_id, add instance_type
  - remove topic, it was unused
- 1.2 - Remove topic from run_instance, it was unused
- 1.3 - Remove instance_id, add instance to live_migration
- 1.4 - Remove update_db from prep_resize
- 1.5 - Add reservations argument to prep_resize()
- 1.6 - Remove reservations argument to run_instance()
- 1.7 - Add create_volume() method, remove topic from live_migration()
- 2.0 - Remove 1.x backwards compat
- 2.1 - Add image_id to create_volume()
- 2.2 - Remove reservations argument to create_volume()
- 2.3 - Remove create_volume()
- **2.4 - Change update_service_capabilities()**
  - accepts a list of capabilities
- 2.5 - Add get_backdoor_port()

•2.6 - Add select_hosts()

... Grizzly supports message version 2.6. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.6.

•2.7 - Add select_destinations()

•**2.8 - Deprecate prep_resize() – JUST KIDDING. It is still used** by the compute manager for retries.

•2.9 - Added the legacy_bdm_in_spec parameter to run_instance()

... Havana supports message version 2.9. So, any changes to existing methods in 2.x after that point should be done such that they can handle the version_cap being set to 2.9.

•Deprecated live_migration() call, moved to conductor

•Deprecated select_hosts()

3.0 - Removed backwards compat

... Icehouse and Juno support message version 3.0. So, any changes to existing methods in 3.x after that point should be done such that they can handle the version_cap being set to 3.0.

•3.1 - Made select_destinations() send flavor object

•4.0 - Removed backwards compat for Icehouse

•4.1 - Add update_aggregates() and delete_aggregate()

•**4.2 - Added update_instance_info(), delete_instance_info(), and** sync_instance_info() methods

... Kilo support message version 4.2. So, any changes to existing methods in 4.x after that point should be done such that they can handle the version_cap being set to 4.2.

VERSION_ALIASES = {'kilo': '4.2', 'grizzly': '2.6', 'havana': '2.9', 'juno': '3.0', 'icehouse': '3.0'}

delete_aggregate(*ctxt*, *aggregate*)

delete_instance_info(*ctxt*, *host_name*, *instance_uuid*)

select_destinations(*ctxt*, *request_spec*, *filter_properties*)

sync_instance_info(*ctxt*, *host_name*, *instance_uuids*)

update_aggregates(*ctxt*, *aggregates*)

update_instance_info(*ctxt*, *host_name*, *instance_info*)

3.7.613 The nova.scheduler.scheduler_options Module

SchedulerOptions monitors a local .json file for changes and loads it if needed. This file is converted to a data structure and passed into the filtering and weighing functions which can use it for dynamic configuration.

class SchedulerOptions

Bases: object

SchedulerOptions monitors a local .json file for changes and loads it if needed. This file is converted to a data structure and passed into the filtering and weighing functions which can use it for dynamic configuration.

get_configuration(*filename=None*)

Check the json file for changes and load it if needed.

3.7.614 The nova.scheduler.utils Module

Utility methods for scheduling.

class GroupDetails
Bases: tuple

GroupDetails(hosts, policies)

hosts
Alias for field number 0

policies
Alias for field number 1

build_request_spec(*ctxt*, *image*, *instances*, *instance_type=None*)
Build a request_spec for the scheduler.

The request_spec assumes that all instances to be scheduled are the same type.

parse_options(*opts*, *sep='='*, *converter=<type 'str'>*, *name=''*)
Parse a list of options, each in the format of <key><sep><value>. Also use the converter to convert the value into desired type.

Params opts list of options, e.g. from oslo_config.cfg.ListOpt

Params sep the separator

Params converter callable object to convert the value, should raise ValueError for conversion failure

Params name name of the option

Returns a lists of tuple of values (key, converted_value)

populate_filter_properties(*filter_properties*, *host_state*)
Add additional information to the filter properties after a node has been selected by the scheduling process.

populate_retry(*filter_properties*, *instance_uuid*)

retry_on_timeout(*retries=1*)
Retry the call in case a MessagingTimeout is raised.

A decorator for retrying calls when a service dies mid-request.

Parameters **retries** – Number of retries

Returns Decorator

retry_select_destinations(*func*)

set_vm_state_and_notify(*context*, *instance_uuid*, *service*, *method*, *updates*, *ex*, *request_spec*, *db*)
changes VM state and notifies.

setup_instance_group(*context*, *request_spec*, *filter_properties*)
Add group_hosts and group_policies fields to filter_properties dict based on instance uuids provided in request_spec, if those instances are belonging to a group.

Parameters

- **request_spec** – Request spec
- **filter_properties** – Filter properties

validate_filter(*filter*)
Validates that the filter is configured in the default filters.

3.7.615 The nova.scheduler.weights.io_ops Module

Io Ops Weigher. Weigh hosts by their io ops number.

The default is to preferably choose light workload compute hosts. If you prefer choosing heavy workload compute hosts, you can set 'io_ops_weight_multiplier' option to a positive number and the weighing has the opposite effect of the default.

class IoOpsWeigher
Bases: nova.scheduler.weights.BaseHostWeigher

minval = 0

weight_multiplier()
Override the weight multiplier.

3.7.616 The nova.scheduler.weights.metrics Module

Metrics Weigher. Weigh hosts by their metrics.

This weigher can compute the weight based on the compute node host's various metrics. The to-be weighed metrics and their weighing ratio are specified in the configuration file as the followings:

[metrics] weight_setting = name1=1.0, name2=-1.0

The final weight would be name1.value * 1.0 + name2.value * -1.0.

class MetricsWeigher
Bases: nova.scheduler.weights.BaseHostWeigher

weight_multiplier()
Override the weight multiplier.

3.7.617 The nova.scheduler.weights.ram Module

RAM Weigher. Weigh hosts by their RAM usage.

The default is to spread instances across all hosts evenly. If you prefer stacking, you can set the 'ram_weight_multiplier' option to a negative number and the weighing has the opposite effect of the default.

class RAMWeigher
Bases: nova.scheduler.weights.BaseHostWeigher

minval = 0

weight_multiplier()
Override the weight multiplier.

3.7.618 The nova.service Module

Generic Node base class for all workers that run on hosts.

class Service(*host*, *binary*, *topic*, *manager*, *report_interval=None*, *periodic_enable=None*, *periodic_fuzzy_delay=None*, *periodic_interval_max=None*, *db_allowed=True*, **args*, ***kwargs*)
Bases: nova.openstack.common.service.Service

Service object for binaries running on hosts.

A service takes a manager and enables rpc by listening to queues based on topic. It also periodically runs tasks on the manager and reports its state to the database services table.

basic_config_check()
Perform basic config checks before starting processing.

classmethod create(*host=None*, *binary=None*, *topic=None*, *manager=None*, *report_interval=None*, *periodic_enable=None*, *periodic_fuzzy_delay=None*, *periodic_interval_max=None*, *db_allowed=True*)
Instantiates class and passes back application object.

Parameters

- **host** – defaults to CONF.host
- **binary** – defaults to basename of executable
- **topic** – defaults to bin_name - 'nova-' part
- **manager** – defaults to CONF.<topic>_manager
- **report_interval** – defaults to CONF.report_interval
- **periodic_enable** – defaults to CONF.periodic_enable
- **periodic_fuzzy_delay** – defaults to CONF.periodic_fuzzy_delay
- **periodic_interval_max** – if set, the max time to wait between runs

kill()
Destroy the service object in the datastore.

periodic_tasks(*raise_on_error=False*)
Tasks to be run at a periodic interval.

start()

stop()

class WSGIService(*name*, *loader=None*, *use_ssl=False*, *max_url_len=None*)
Bases: `object`

Provides ability to launch API from a 'paste' configuration.

reset()
Reset server greenpool size to default.

Returns None

start()
Start serving this service using loaded configuration.

Also, retrieve updated port number in case '0' was passed in, which indicates a random port should be used.

Returns None

stop()
Stop serving this API.

Returns None

wait()
Wait for the service to stop serving this API.

Returns None

process_launcher()

serve(*server*, *workers=None*)

wait()

3.7.619 The nova.servicegroup.api Module

Define APIs for the servicegroup access.

class API(**args*, ***kwargs*)
Bases: object

get_all(*group_id*)
Returns ALL members of the given group.

join(*member*, *group*, *service=None*)
Add a new member to a service group.

Parameters

- **member** – the joined member ID/name
- **group** – the group ID/name, of the joined member
- **service** – a *nova.service.Service* object

service_is_up(*member*)
Check if the given member is up.

3.7.620 The nova.servicegroup.drivers.base Module

class Driver
Bases: object

Base class for all ServiceGroup drivers.

is_up(*member*)
Check whether the given member is up.

join(*member*, *group*, *service=None*)
Add a new member to a service group.

Parameters

- **member** – the joined member ID/name
- **group** – the group ID/name, of the joined member
- **service** – a *nova.service.Service* object

3.7.621 The nova.servicegroup.drivers.db Module

class DbDriver(**args*, ***kwargs*)
Bases: nova.servicegroup.drivers.base.Driver

is_up(*service_ref*)
Moved from nova.utils Check whether a service is up based on last heartbeat.

join(*member*, *group*, *service=None*)
Add a new member to a service group.

Parameters

- **member** – the joined member ID/name

- **group** – the group ID/name, of the joined member
- **service** – a *nova.service.Service* object

3.7.622 The nova.servicegroup.drivers.mc Module

class MemcachedDriver(**args*, ***kwargs*)
Bases: nova.servicegroup.drivers.base.Driver

is_up(*service_ref*)
Moved from nova.utils Check whether a service is up based on last heartbeat.

join(*member_id*, *group_id*, *service=None*)
Join the given service with its group.

3.7.623 The nova.servicegroup.drivers.zk Module

class ZooKeeperDriver(**args*, ***kwargs*)
Bases: nova.servicegroup.drivers.base.Driver

ZooKeeper driver for the service group API.

is_up(*service_ref*)

join(*member*, *group*, *service=None*)
Add a new member to a service group.

Parameters

- **member** – the joined member ID/name
- **group** – the group ID/name, of the joined member
- **service** – a *nova.service.Service* object

3.7.624 The nova.storage.linuxscsi Module

Generic linux scsi subsystem utilities.

echo_scsi_command(*path*, *content*)
Used to echo strings to scsi subsystem.

find_multipath_device(*device*)
Try and discover the multipath device for a volume.

get_device_info(*device*)

get_device_list()

remove_device(*device*)

rescan_hosts(*hbas*)

3.7.625 The nova.test Module

Base classes for our unit tests.

Allows overriding of flags for use of fakes, and some black magic for inline callbacks.

class APICoverage
Bases: object

cover_api = None

test_api_methods()

class BaseHookTestCase(**args*, ***kwargs*)
Bases: nova.test.NoDBTestCase

assert_has_hook(*expected_name*, *func*)

class MatchType(*wanttype*)
Bases: object

Matches any instance of a specified type

The MatchType class is a helper for use with the mock.assert_called_with() method that lets you assert that a particular parameter has a specific data type. It enables strict check than the built in mock.ANY helper, and is the equivalent of the mox.IsA() function from the legacy mox library

Example usage could be:

mock_some_method.assert_called_once_with("hello", MatchType(objects.Instance), mock.ANY, "world", MatchType(objects.KeyPair))

class NoDBTestCase(**args*, ***kwargs*)
Bases: nova.test.TestCase

NoDBTestCase differs from TestCase in that DB access is not supported. This makes tests run significantly faster. If possible, all new tests should derive from this class.

USES_DB = False

class NullHandler(*level=0*)
Bases: logging.Handler

custom default NullHandler to attempt to format the record.

Used in conjunction with log_fixture.get_logging_handle_error_fixture to detect formatting errors in debug level logs without saving the logs.

createLock()

emit(*record*)

handle(*record*)

class SampleNetworks(*host=None*)
Bases: fixtures.fixture.Fixture

Create sample networks in the database.

setUp()

class TestCase(**args*, ***kwargs*)
Bases: testtools.testcase.TestCase

Test case base class for all unit tests.

Due to the slowness of DB access, please consider deriving from *NoDBTestCase* first.

REQUIRES_LOCKING = False

TIMEOUT_SCALING_FACTOR = 1

USES_DB = True

assertJsonEqual(*expected*, *observed*)

assertPublicAPISignatures(*baseinst*, *inst*)

flags(***kw*)
Override flag variables for a test.

setUp()
Run before each test method to initialize test environment.

start_service(*name*, *host=None*, ***kwargs*)

exception TestingException
Bases: exceptions.Exception

class TimeOverride
Bases: fixtures.fixture.Fixture

Fixture to start and remove time override.

setUp()

class skipIf(*condition*, *reason*)
Bases: object

3.7.626 The nova.utils Module

Utilities and helper functions.

class ExceptionHelper(*target*)
Bases: object

Class to wrap another and translate the ClientExceptions raised by its function calls to the actual ones.

class UndoManager
Bases: object

Provides a mechanism to facilitate rolling back a series of actions when an exception is raised.

rollback_and_reraise(*msg=None*, ***kwargs*)
Rollback a series of actions then re-raise the exception.

Note: (sirp) This should only be called within an exception handler.

---

undo_with(*undo_func*)

check_isinstance(*obj*, *cls*)
Checks that obj is of type cls, and lets PyLint infer types.

check_string_length(*value*, *name=None*, *min_length=0*, *max_length=None*)
Check the length of specified string :param value: the value of the string :param name: the name of the string :param min_length: the min_length of the string :param max_length: the max_length of the string

convert_to_list_dict(*lst*, *label*)
Convert a value or list into a list of dicts.

convert_version_to_int(*version*)

convert_version_to_str(*version_int*)

convert_version_to_tuple(*version_str*)

dict_to_metadata(*metadata*)

execute(**cmd*, ***kwargs*)
Convenience wrapper around oslo's execute() method.

expects_func_args(**args*)

filter_and_format_resource_metadata(*resource_type*, *resource_list*, *search_filts*, *metadata_type=None*)
Get all metadata for a list of resources after filtering.

Search_filts is a list of dictionaries, where the values in the dictionary can be string or regex string, or a list of strings/regex strings.

Let's call a dict a 'filter block' and an item in the dict a 'filter'. A tag is returned if it matches ALL the filters in a filter block. If more than one values are specified for a filter, a tag is returned if it matches ATLEAST ONE value of the filter. If more than one filter blocks are specified, the tag should match ALL the filter blocks.

For example:

search_filts = [{'key': ['key1', 'key2'], 'value': 'val1'}, {'value': 'val2'}]

The filter translates to 'match any tag for which':

((key=key1 AND value=val1) OR (key=key2 AND value=val1)) AND (value=val2)

This example filter will never match a tag.

param resource_type The resource type as a string, e.g. 'instance'

param resource_list List of resource objects

param search_filts Filters to filter metadata to be returned. Can be dict (e.g. {'key': 'env', 'value': 'prod'}, or a list of dicts (e.g. [{'key': 'env'}, {'value': 'beta'}]. Note that the values of the dict can be regular expressions.

param metadata_type Provided to search for a specific metadata type (e.g. 'system_metadata')

returns List of dicts where each dict is of the form {'key': 'somekey', 'value': 'somevalue', 'instance_id': 'some-instance-uuid-aaa'} if resource_type is 'instance'.

generate_mac_address()
Generate an Ethernet MAC address.

generate_password(*length=None*, *symbolgroups=('23456789'*, *'ABCDEFGHJKLMNPQRSTUVWXYZ'*, *'abcdefghijkmnopqrstuvwxyz')*)
Generate a random password from the supplied symbol groups.

At least one symbol from each group will be included. Unpredictable results if length is less than the number of symbol groups.

Believed to be reasonably secure (with a reasonable password length!)

generate_uid(*topic*, *size=8*)

get_auto_disk_config_from_image_props(*image_properties*)

get_auto_disk_config_from_instance(*instance=None*, *sys_meta=None*)

get_hash_str(*base_str*)
returns string that represents hash of base_str (in hex format).

get_image_from_system_metadata(*system_meta*)

get_image_metadata_from_volume(*volume*)

get_ip_version(*network*)
Returns the IP version of a network (IPv4 or IPv6).

Raises AddrFormatError if invalid network.

get_my_linklocal(*interface*)

get_shortened_ipv6(*address*)

get_shortened_ipv6_cidr(*address*)

get_system_metadata_from_image(*image_meta*, *flavor=None*)

get_wrapped_function(*function*)
Get the method at the bottom of a stack of decorators.

instance_meta(*instance*)

instance_sys_meta(*instance*)

is_auto_disk_config_disabled(*auto_disk_config_raw*)

is_neutron()

is_none_string(*val*)
Check if a string represents a None value.

is_valid_cidr(*address*)
Check if address is valid

The provided address can be a IPv6 or a IPv4 CIDR address.

is_valid_ipv6_cidr(*address*)

last_bytes(*file_like_object*, *num*)
Return num bytes from the end of the file, and remaining byte count.

Parameters

- **file_like_object** – The file to read
- **num** – The number of bytes to return

:returns (data, remaining)

last_completed_audit_period(*unit=None*, *before=None*)
This method gives you the most recently *completed* audit period.

arguments:

units: string, one of 'hour', 'day', 'month', 'year' Periods normally begin at the beginning (UTC) of the period unit (So a 'day' period begins at midnight UTC, a 'month' unit on the 1st, a 'year' on Jan, 1) unit string may be appended with an optional offset like so: 'day@18' This will begin the period at 18:00 UTC. 'month@15' starts a monthly period on the 15th, and year@3 begins a yearly one on March 1st.

before: Give the audit period most recently completed before <timestamp>. Defaults to now.

returns: 2 tuple of datetimes (begin, end) The begin timestamp of this audit period is the same as the end of the previous.

make_dev_path(*dev*, *partition=None*, *base='/dev'*)
Return a path to a particular device.

```
>>> make_dev_path('xvdc')
/dev/xvdc
```

```
>>> make_dev_path('xvdc', 1)
/dev/xvdc1
```

metadata_to_dict (*metadata*, *filter_deleted=False*)

mkfs (*fs*, *path*, *label=None*, *run_as_root=False*)
Format a file or block device

Parameters

- **fs** – Filesystem type (examples include 'swap', 'ext3', 'ext4' 'btrfs', etc.)
- **path** – Path to file or block device to format
- **label** – Volume label to use

monkey_patch ()
If the CONF.monkey_patch set as True, this function patches a decorator for all functions in specified modules. You can set decorators for each modules using CONF.monkey_patch_modules. The format is "Module path:Decorator function". Example: 'nova.api.ec2.cloud:nova.notifications.notify_decorator'

Parameters of the decorator is as follows. (See nova.notifications.notify_decorator)

name - name of the function function - object of the function

novadir ()

parse_server_string (*server_str*)
Parses the given server_string and returns a tuple of host and port. If it's not a combination of host part and port, the port element is an empty string. If the input is invalid expression, return a tuple of two empty strings.

read_cached_file (*filename*, *cache_info*, *reload_func=None*)
Read from a file if it has been modified.

Parameters

- **cache_info** – dictionary to hold opaque cache.
- **reload_func** – optional function to be called with data when file is reloaded due to a modification.

Returns data from file

read_file_as_root (*file_path*)
Secure helper to read file as root.

safe_ip_format (*ip*)
Transform ip string to "safe" format.

Will return ipv4 addresses unchanged, but will nest ipv6 addresses inside square brackets.

safe_truncate (*value*, *length*)
Safely truncates unicode strings such that their encoded length is no greater than the length provided.

sanitize_hostname (*hostname*)
Return a hostname which conforms to RFC-952 and RFC-1123 specs.

spawn (*func*, **args*, ***kwargs*)
Passthrough method for eventlet.spawn.

This utility exists so that it can be stubbed for testing without interfering with the service spawns.

It will also grab the context from the threadlocal store and add it to the store on the new thread. This allows for continuity in logging the context when using this method to spawn a new thread.

spawn_n (*func*, **args*, ***kwargs*)
Passthrough method for eventlet.spawn_n.

This utility exists so that it can be stubbed for testing without interfering with the service spawns.

It will also grab the context from the threadlocal store and add it to the store on the new thread. This allows for continuity in logging the context when using this method to spawn a new thread.

ssh_execute (*dest*, **cmd*, ***kwargs*)
Convenience wrapper to execute ssh command.

tempdir (**args*, ***kwds*)

temporary_chown (**args*, ***kwds*)
Temporarily chown a path.

Parameters **owner_uid** – UID of temporary owner (defaults to current user)

temporary_mutation (**args*, ***kwds*)
Temporarily set the attr on a particular object to a given value then revert when finished.

One use of this is to temporarily set the read_deleted flag on a context object:

with temporary_mutation(context, read_deleted="yes"): do_something_that_needed_deleted_objects()

trycmd (**args*, ***kwargs*)
Convenience wrapper around oslo's trycmd() method.

utf8 (*value*)
Try to turn a string into utf-8 if possible.

Code is directly from the utf8 function in http://github.com/facebook/tornado/blob/master/tornado/escape.py

utils_opts = [<oslo_config.cfg.IntOpt object at 0x7f858be3b410>, <oslo_config.cfg.StrOpt object at 0x7f858be3bad0>, <oslo_
This group is for very specific reasons.

If you're:

- Working around an issue in a system tool (e.g. libvirt or qemu) where the fix is in flight/discussed in that community.
- The tool can be/is fixed in some distributions and rather than patch the code those distributions can trivially set a config option to get the "correct" behavior.

This is a good place for your workaround.

Please use with care! Document the BugID that your workaround is paired with.

validate_integer (*value*, *name*, *min_value=None*, *max_value=None*)
Make sure that value is a valid integer, potentially within range.

vpn_ping (*address*, *port*, *timeout=0.05*, *session_id=None*)
Sends a vpn negotiation packet and returns the server session.

Returns Boolean indicating whether the vpn_server is listening. Basic packet structure is below.

Client packet (14 bytes):

```
 0 1      8 9  13
+-+--------+-----+
|x| cli_id |?????|
+-+--------+-----+
x = packet identifier 0x38
cli_id = 64 bit identifier
? = unknown, probably flags/padding
```

Server packet (26 bytes):

```
 0 1      8 9  13 14    21 2225
+-+--------+-----+--------+----+
|x| srv_id |?????| cli_id |????|
+-+--------+-----+--------+----+
x = packet identifier 0x40
cli_id = 64 bit identifier
? = unknown, probably flags/padding
bit 9 was 1 and the rest were 0 in testing
```

walk_class_hierarchy(*clazz*, *encountered=None*)
Walk class hierarchy, yielding most derived classes first.

xhtml_escape(*value*)
Escapes a string so it is valid within XML or XHTML.

3.7.627 The nova.version Module

package_string()

product_string()

vendor_string()

version_string_with_package()

3.7.628 The nova.virt.block_device Module

class **DriverBlankBlockDevice**(*bdm*)
Bases: nova.virt.block_device.DriverVolumeBlockDevice

attach(*context*, *instance*, *volume_api*, *virt_driver*, *wait_func=None*, *do_check_attach=True*)

class **DriverBlockDevice**(*bdm*)
Bases: dict

A dict subclass that represents block devices used by the virt layer.

Uses block device objects internally to do the database access.

_fields and _legacy_fields class attributes present a set of fields that are expected on a certain DriverBlockDevice type. We may have more legacy versions in the future.

If an attribute access is attempted for a name that is found in the _proxy_as_attr set, it will be proxied to the underlying object. This allows us to access stuff that is not part of the data model that all drivers understand.

The save() method allows us to update the database using the underlying object. _update_on_save class attribute dictionary keeps the following mapping:

{'object field name': 'driver dict field name (or None if same)'}

These fields will be updated on the internal object, from the values in the dict, before the actual database update is done.

attach(***kwargs*)
Make the device available to be used by VMs.

To be overridden in subclasses with the connecting logic for the type of device the subclass represents.

legacy()
Basic legacy transformation.

Basic method will just drop the fields that are not in _legacy_fields set. Override this in subclass if needed.

save()

class **DriverEphemeralBlockDevice**(*bdm*)
Bases: nova.virt.block_device.DriverBlockDevice

legacy(*num=0*)

class **DriverImageBlockDevice**(*bdm*)
Bases: nova.virt.block_device.DriverVolumeBlockDevice

attach(*context*, *instance*, *volume_api*, *virt_driver*, *wait_func=None*, *do_check_attach=True*)

class **DriverSnapshotBlockDevice**(*bdm*)
Bases: nova.virt.block_device.DriverVolumeBlockDevice

attach(*context*, *instance*, *volume_api*, *virt_driver*, *wait_func=None*, *do_check_attach=True*)

class **DriverSwapBlockDevice**(*bdm*)
Bases: nova.virt.block_device.DriverBlockDevice

class **DriverVolumeBlockDevice**(*bdm*)
Bases: nova.virt.block_device.DriverBlockDevice

attach(*obj*, *context*, **args*, ***kwargs*)

refresh_connection_info(*obj*, *context*, **args*, ***kwargs*)

save()

attach_block_devices(*block_device_mapping*, **attach_args*, ***attach_kwargs*)

convert_all_volumes(**volume_bdms*)

convert_volume(*volume_bdm*)

get_swap(*transformed_list*)
Get the swap device out of the list context.

The block_device_info needs swap to be a single device, not a list - otherwise this is a no-op.

is_block_device_mapping(*bdm*)

is_implemented(*bdm*)

legacy_block_devices(*block_device_mapping*)

refresh_conn_infos(*block_device_mapping*, **refresh_args*, ***refresh_kwargs*)

update_db(*method*)

3.7.629 The nova.virt.configdrive Module

Config Drive v2 helper.

class **ConfigDriveBuilder**(*instance_md=None*)
Bases: object

Build config drives, optionally as a context manager.

add_instance_metadata(*instance_md*)

cleanup()

make_drive(*path*, *image_type='raw'*)
Make the config drive.

Parameters

- **path** – the path to place the config drive image at
- **image_type** – host side image format

:raises ProcessExecuteError if a helper process has failed.

required_by(*instance*)

update_instance(*instance*)
Update the instance config_drive setting if necessary

The image or configuration file settings may override the default instance setting. In this case the instance needs to mirror the actual virtual machine configuration.

3.7.630 The nova.virt.diagnostics Module

class CpuDiagnostics(*time=0*)
Bases: object

class Diagnostics(*state=None*, *driver=None*, *hypervisor_os=None*, *uptime=0*, *cpu_details=None*, *nic_details=None*, *disk_details=None*, *config_drive=False*)
Bases: object

add_cpu(*time=0*)

add_disk(*id=''*, *read_bytes=0*, *read_requests=0*, *write_bytes=0*, *write_requests=0*, *errors_count=0*)

add_nic(*mac_address='00:00:00:00:00:00'*, *rx_octets=0*, *rx_errors=0*, *rx_drop=0*, *rx_packets=0*, *tx_octets=0*, *tx_errors=0*, *tx_drop=0*, *tx_packets=0*)

serialize()

version = '1.0'

class DiskDiagnostics(*id=''*, *read_bytes=0*, *read_requests=0*, *write_bytes=0*, *write_requests=0*, *errors_count=0*)
Bases: object

class MemoryDiagnostics(*maximum=0*, *used=0*)
Bases: object

class NicDiagnostics(*mac_address='00:00:00:00:00:00'*, *rx_octets=0*, *rx_errors=0*, *rx_drop=0*, *rx_packets=0*, *tx_octets=0*, *tx_errors=0*, *tx_drop=0*, *tx_packets=0*)
Bases: object

3.7.631 The nova.virt.disk.api Module

Utility methods to resize, repartition, and modify disk images.

Includes injection of SSH PGP keys into authorized_keys file.

can_resize_image(*image*, *size*)
Check whether we can resize the container image file. :param image: path to local image file :param size: the image size in bytes

clean_lxc_namespace(*container_dir*)
Clean up the container namespace rootfs mounting one spawned.

It will umount the mounted names that are mounted but leave the linked devices alone.

extend(*image*, *size*)
Increase image to size.

Parameters

- **image** – instance of nova.virt.image.model.Image
- **size** – image size in bytes

get_disk_size(*path*)
Get the (virtual) size of a disk image

Parameters **path** – Path to the disk image

Returns Size (in bytes) of the given disk image as it would be seen by a virtual machine.

get_file_extension_for_os_type(*os_type*, *specified_fs=None*)

get_fs_type_for_os_type(*os_type*)

inject_data(*image*, *key=None*, *net=None*, *metadata=None*, *admin_password=None*, *files=None*, *partition=None*, *mandatory=()*)
Inject the specified items into a disk image.

Parameters

- **image** – instance of nova.virt.image.model.Image
- **key** – the SSH public key to inject
- **net** – the network configuration to inject
- **metadata** – the user metadata to inject
- **admin_password** – the root password to set
- **files** – the files to copy into the image
- **partition** – the partition number to access
- **mandatory** – the list of parameters which must not fail to inject

If an item name is not specified in the MANDATORY iterable, then a warning is logged on failure to inject that item, rather than raising an exception.

it will mount the image as a fully partitioned disk and attempt to inject into the specified partition number.

If PARTITION is not specified the image is mounted as a single partition.

Returns True if all requested operations completed without issue. Raises an exception if a mandatory item can't be injected.

inject_data_into_fs(*fs*, *key*, *net*, *metadata*, *admin_password*, *files*, *mandatory=()*)
Injects data into a filesystem already mounted by the caller. Virt connections can call this directly if they mount their fs in a different way to inject_data.

If an item name is not specified in the MANDATORY iterable, then a warning is logged on failure to inject that item, rather than raising an exception.

Returns True if all requested operations completed without issue. Raises an exception if a mandatory item can't be injected.

is_image_extendable(*image*)
Check whether we can extend the image.

mkfs(*os_type*, *fs_label*, *target*, *run_as_root=True*, *specified_fs=None*)
Format a file or block device using a user provided command for each os type. If user has not provided any configuration, format type will be used according to a default_ephemeral_format configuration or a system defaults.

resize2fs(*image*, *check_exit_code=False*, *run_as_root=False*)

setup_container(*image*, *container_dir*)
Setup the LXC container.

Parameters

- **image** – instance of nova.virt.image.model.Image
- **container_dir** – directory to mount the image at

It will mount the loopback image to the container directory in order to create the root filesystem for the container.

Returns path of image device which is mounted to the container directory.

teardown_container(*container_dir*, *container_root_device=None*)
Teardown the container rootfs mounting once it is spawned.

It will umount the container that is mounted, and delete any linked devices.

3.7.632 The nova.virt.disk.mount.api Module

Support for mounting virtual image files.

class Mount(*image*, *mount_dir*, *partition=None*, *device=None*)
Bases: object

Standard mounting operations, that can be overridden by subclasses.

The basic device operations provided are get, map and mount, to be called in that order.

do_mount()
Call the get, map and mnt operations.

do_teardown()
Call the umnt, unmap, and unget operations.

do_umount()
Call the unmnt operation.

flush_dev()

get_dev()
Make the image available as a block device in the file system.

static instance_for_device(*image*, *mountdir*, *partition*, *device*)
Get a Mount instance for the device type

Parameters

- **image** – instance of nova.virt.image.model.Image
- **mountdir** – path to mount the image at
- **partition** – partition number to mount
- **device** – mounted device path

static instance_for_format(*image*, *mountdir*, *partition*)
Get a Mount instance for the image type

Parameters

- **image** – instance of nova.virt.image.model.Image
- **mountdir** – path to mount the image at
- **partition** – partition number to mount

map_dev()
Map partitions of the device to the file system namespace.

mnt_dev()
Mount the device into the file system.

mode = None

reset_dev()
Reset device paths to allow unmounting.

unget_dev()
Release the block device from the file system namespace.

unmap_dev()
Remove partitions of the device from the file system namespace.

unmnt_dev()
Unmount the device from the file system.

3.7.633 The nova.virt.disk.mount.loop Module

Support for mounting images with the loop device.

class LoopMount(*image*, *mount_dir*, *partition=None*, *device=None*)
Bases: nova.virt.disk.mount.api.Mount

loop back support for raw images.

get_dev()

mode = 'loop'

unget_dev()

3.7.634 The nova.virt.disk.mount.nbd Module

Support for mounting images with qemu-nbd.

class NbdMount(*image*, *mount_dir*, *partition=None*, *device=None*)
Bases: nova.virt.disk.mount.api.Mount

qemu-nbd support disk images.

flush_dev()
flush NBD block device buffer.

get_dev()
Retry requests for NBD devices.

mode = 'nbd'

unget_dev()

3.7.635 The nova.virt.disk.vfs.api Module

class VFS(*image*, *partition*)
Bases: object

Interface for manipulating disk image.

The VFS class defines an interface for manipulating files within a virtual disk image filesystem. This allows file injection code to avoid the assumption that the virtual disk image can be mounted in the host filesystem.

All paths provided to the APIs in this class should be relative to the root of the virtual disk image filesystem. Subclasses will translate paths as required by their implementation.

append_file(*path*, *content*)
Appends @content to the end of the file.

Append @content to the end of the file identified by @path, creating the file if it does not already exist.

get_image_fs()
Returns the filesystem type or an empty string.

Determine the filesystem type whether the disk image is partition less.

guestfs_ready = False

has_file(*path*)
Returns a True if the file identified by @path exists.

static instance_for_image(*image*, *partition*)
Get a VFS instance for the image

Parameters

- **image** – instance of nova.virt.image.model.Image
- **partition** – the partition number to access

make_path(*path*)
Creates a directory @path.

Create a directory @path, including all intermedia path components if they do not already exist.

read_file(*path*)
Returns the entire contents of the file identified by @path.

replace_file(*path*, *content*)
Replaces contents of the file.

Replace the entire contents of the file identified by @path, with @content, creating the file if it does not already exist.

set_ownership(*path*, *user*, *group*)
Sets the ownership on the file.

Set the ownership on the file identified by @path to the username @user and groupname @group. Either of @user or @group may be None, in which case the current ownership will be left unchanged. The ownership must be passed in string form, allowing subclasses to translate to uid/gid form as required. The file must exist prior to this call.

set_permissions(*path*, *mode*)
Sets the permissions on the file.

Set the permissions on the file identified by @path to @mode. The file must exist prior to this call.

setup (*mount=True*)
Performs any one-time setup.

Perform any one-time setup tasks to make the virtual filesystem available to future API calls.

teardown ()
Releases all resources initialized in the setup method.

3.7.636 The nova.virt.disk.vfs.guestfs Module

class VFSGuestFS (*image*, *partition=None*)
Bases: nova.virt.disk.vfs.api.VFS

This class implements a VFS module that uses the libguestfs APIs to access the disk image. The disk image is never mapped into the host filesystem, thus avoiding any potential for symlink attacks from the guest filesystem.

append_file (*path*, *content*)

configure_debug ()
Configures guestfs to be verbose.

get_image_fs ()

has_file (*path*)

inspect_capabilities ()
Determines whether guestfs is well configured.

make_path (*path*)

read_file (*path*)

replace_file (*path*, *content*)

set_ownership (*path*, *user*, *group*)

set_permissions (*path*, *mode*)

setup (*mount=True*)

setup_os ()

setup_os_inspect ()

setup_os_root (*root*)

setup_os_static ()

teardown ()

force_tcg (*force=True*)
Prevent libguestfs trying to use KVM acceleration

It is a good idea to call this if it is known that KVM is not desired, even if technically available.

3.7.637 The nova.virt.disk.vfs.localfs Module

class VFSLocalFS (*image*, *partition=None*, *imgdir=None*)
Bases: nova.virt.disk.vfs.api.VFS

os.path.join() with safety check for injected file paths.

Join the supplied path components and make sure that the resulting path we are injecting into is within the mounted guest fs. Trying to be clever and specifying a path with '..' in it will hit this safeguard.

append_file(*path*, *content*)

get_image_fs()

has_file(*path*)

make_path(*path*)

read_file(*path*)

replace_file(*path*, *content*)

set_ownership(*path*, *user*, *group*)

set_permissions(*path*, *mode*)

setup(*mount=True*)

teardown()

3.7.638 The nova.virt.driver Module

Driver base-classes:

(Beginning of) the contract that compute drivers must follow, and shared types that support that contract

class ComputeDriver(*virtapi*)

Bases: object

Base class for compute drivers.

The interface to this class talks in terms of 'instances' (Amazon EC2 and internal Nova terminology), by which we mean 'running virtual machine' (XenAPI terminology) or domain (Xen or libvirt terminology).

An instance has an ID, which is the identifier chosen by Nova to represent the instance further up the stack. This is unfortunately also called a 'name' elsewhere. As far as this layer is concerned, 'instance ID' and 'instance name' are synonyms.

Note that the instance ID or name is not human-readable or customer-controlled – it's an internal ID chosen by Nova. At the nova.virt layer, instances do not have human-readable names at all – such things are only known higher up the stack.

Most virtualization platforms will also have their own identity schemes, to uniquely identify a VM or domain. These IDs must stay internal to the platform-specific layer, and never escape the connection interface. The platform-specific layer is responsible for keeping track of which instance ID maps to which platform-specific ID, and vice versa.

Some methods here take an instance of nova.compute.service.Instance. This is the data structure used by nova.compute to store details regarding an instance, and pass them into this layer. This layer is responsible for translating that generic data structure into terms that are specific to the virtualization platform.

add_to_aggregate(*context*, *aggregate*, *host*, ***kwargs*)

Add a compute host to an aggregate.

attach_interface(*instance*, *image_meta*, *vif*)

Attach an interface to the instance.

Parameters **instance** – nova.objects.instance.Instance

attach_volume(*context*, *connection_info*, *instance*, *mountpoint*, *disk_bus=None*, *device_type=None*, *encryption=None*)
Attach the disk to the instance at mountpoint using info.

block_stats(*instance*, *disk_id*)
Return performance counters associated with the given disk_id on the given instance. These are returned as [rd_req, rd_bytes, wr_req, wr_bytes, errs], where rd indicates read, wr indicates write, req is the total number of I/O requests made, bytes is the total number of bytes transferred, and errs is the number of requests held up due to a full pipeline.

All counters are long integers.

This method is optional. On some platforms (e.g. XenAPI) performance statistics can be retrieved directly in aggregate form, without Nova having to do the aggregation. On those platforms, this method is unused.

Note that this function takes an instance ID.

capabilities = {'supports_recreate': False, 'has_imagecache': False, 'supports_migrate_to_same_host': False}

change_instance_metadata(*context*, *instance*, *diff*)
Applies a diff to the instance metadata.

This is an optional driver method which is used to publish changes to the instance's metadata to the hypervisor. If the hypervisor has no means of publishing the instance metadata to the instance, then this method should not be implemented.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance

check_can_live_migrate_destination(*context*, *instance*, *src_compute_info*, *dst_compute_info*, *block_migration=False*, *disk_over_commit=False*)
Check if it is possible to execute live migration.

This runs checks on the destination host, and then calls back to the source host to check the results.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance
- **src_compute_info** – Info about the sending machine
- **dst_compute_info** – Info about the receiving machine
- **block_migration** – if true, prepare for block migration
- **disk_over_commit** – if true, allow disk over commit

Returns a dict containing migration info (hypervisor-dependent)

check_can_live_migrate_destination_cleanup(*context*, *dest_check_data*)
Do required cleanup on dest host after check_can_live_migrate calls

Parameters

- **context** – security context
- **dest_check_data** – result of check_can_live_migrate_destination

check_can_live_migrate_source(*context*, *instance*, *dest_check_data*, *block_device_info=None*)
Check if it is possible to execute live migration.

This checks if the live migration can succeed, based on the results from check_can_live_migrate_destination.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance
- **dest_check_data** – result of check_can_live_migrate_destination
- **block_device_info** – result of _get_instance_block_device_info

Returns a dict containing migration info (hypervisor-dependent)

check_instance_shared_storage_cleanup(*context*, *data*)
Do cleanup on host after check_instance_shared_storage calls

Parameters

- **context** – security context
- **data** – result of check_instance_shared_storage_local

check_instance_shared_storage_local(*context*, *instance*)
Check if instance files located on shared storage.

This runs check on the destination host, and then calls back to the source host to check the results.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance object

check_instance_shared_storage_remote(*context*, *data*)
Check if instance files located on shared storage.

Parameters

- **context** – security context
- **data** – result of check_instance_shared_storage_local

cleanup(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*, *destroy_vifs=True*)
Cleanup the instance resources .

Instance should have been destroyed from the Hypervisor before calling this method.

Parameters

- **context** – security context
- **instance** – Instance object as returned by DB layer.
- **network_info** – `get_instance_nw_info()`
- **block_device_info** – Information about block devices that should be detached from the instance.
- **destroy_disks** – Indicates if disks should be destroyed
- **migrate_data** – implementation specific params

cleanup_host(*host*)
Clean up anything that is necessary for the driver gracefully stop, including ending remote sessions. This is optional.

confirm_migration(*migration*, *instance*, *network_info*)
Confirms a resize, destroying the source VM.

Parameters **instance** – nova.objects.instance.Instance

deallocate_networks_on_reschedule(*instance*)
Does the driver want networks deallocated on reschedule?

default_device_names_for_instance(*instance*, *root_device_name*, **block_device_lists*)
Default the missing device names in the block device mapping.

default_root_device_name(*instance*, *image_meta*, *root_bdm*)
Provide a default root device name for the driver.

delete_instance_files(*instance*)
Delete any lingering instance files for an instance.

Parameters **instance** – nova.objects.instance.Instance

Returns True if the instance was deleted from disk, False otherwise.

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)
Destroy the specified instance from the Hypervisor.

If the instance is not found (for example if networking failed), this function should still succeed. It's probably a good idea to log a warning in that case.

Parameters

- **context** – security context
- **instance** – Instance object as returned by DB layer.
- **network_info** – `get_instance_nw_info()`
- **block_device_info** – Information about block devices that should be detached from the instance.
- **destroy_disks** – Indicates if disks should be destroyed
- **migrate_data** – implementation specific params

detach_interface(*instance*, *vif*)
Detach an interface from the instance.

Parameters **instance** – nova.objects.instance.Instance

detach_volume(*connection_info*, *instance*, *mountpoint*, *encryption=None*)
Detach the disk attached to the instance.

dhcp_options_for_instance(*instance*)
Get DHCP options for this instance.

Some hypervisors (such as bare metal) require that instances boot from the network, and manage their own TFTP service. This requires passing the appropriate options out to the DHCP service. Most hypervisors can use the default implementation which returns None.

This is called during spawn_instance by the compute manager.

Note that the format of the return value is specific to Quantum client API.

Returns

None, or a set of DHCP options, eg:

```
[{'opt_name': 'bootfile-name',
        'opt_value': '/tftpboot/path/to/config'},
    {'opt_name': 'server-ip-address',
        'opt_value': '1.2.3.4'},
    {'opt_name': 'tftp-server',
        'opt_value': '1.2.3.4'}
]
```

emit_event(*event*)
Dispatches an event to the compute manager.

Invokes the event callback registered by the compute manager to dispatch the event. This must only be invoked from a green thread.

ensure_filtering_rules_for_instance(*instance*, *network_info*)
Setting up filtering rules and waiting for its completion.

To migrate an instance, filtering rules to hypervisors and firewalls are inevitable on destination host. (Waiting only for filtering rules to hypervisor, since filtering rules to firewall rules can be set faster).

Concretely, the below method must be called. - setup_basic_filtering (for nova-basic, etc.) - prepare_instance_filter(for nova-instance-instance-xxx, etc.)

to_xml may have to be called since it defines PROJNET, PROJMASK. but libvirt migrates those value through migrateToURI(), so , no need to be called.

Don't use thread for this method since migration should not be started when setting-up filtering rules operations are not completed.

Parameters **instance** – nova.objects.instance.Instance object

estimate_instance_overhead(*instance_info*)
Estimate the virtualization overhead required to build an instance of the given flavor.

Defaults to zero, drivers should override if per-instance overhead calculations are desired.

Parameters **instance_info** – Instance/flavor to calculate overhead for.

Returns Dict of estimated overhead values.

filter_defer_apply_off()
Turn off deferral of IPTables rules and apply the rules now.

filter_defer_apply_on()
Defer application of IPTables rules.

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)
Completes a resize.

Parameters

- **context** – the context for the migration/resize
- **migration** – the migrate/resize information
- **instance** – nova.objects.instance.Instance being migrated/resized
- **disk_info** – the newly transferred disk information
- **network_info** – `get_instance_nw_info()`

- **image_meta** – image object returned by nova.image.glance that defines the image from which this instance was created
- **resize_instance** – True if the instance is being resized, False otherwise
- **block_device_info** – instance volume block device info
- **power_on** – True if the instance should be powered on, False otherwise

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)

Finish reverting a resize.

Parameters

- **context** – the context for the finish_revert_migration
- **instance** – nova.objects.instance.Instance being migrated/resized
- **network_info** – `get_instance_nw_info()`
- **block_device_info** – instance volume block device info
- **power_on** – True if the instance should be powered on, False otherwise

get_all_bw_counters(*instances*)

Return bandwidth usage counters for each interface on each running VM.

Parameters **instances** – nova.objects.instance.InstanceList

get_all_volume_usage(*context*, *compute_host_bdms*)

Return usage info for volumes attached to vms on a given host.-

get_available_nodes(*refresh=False*)

Returns nodenames of all nodes managed by the compute service.

This method is for multi compute-nodes support. If a driver supports multi compute-nodes, this method returns a list of nodenames managed by the service. Otherwise, this method should return [hypervisor_hostname].

get_available_resource(*nodename*)

Retrieve resource information.

This method is called when nova-compute launches, and as part of a periodic task that records the results in the DB.

Parameters **nodename** – node which the caller want to get resources from a driver that manages only one node can safely ignore this

Returns Dictionary describing resources

get_console_output(*context*, *instance*)

Get console output for an instance

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance

get_console_pool_info(*console_type*)

get_device_name_for_instance(*instance*, *bdms*, *block_device_obj*)

Get the next device name based on the block device mapping.

Parameters

- **instance** – nova.objects.instance.Instance that volume is requesting a device name
- **bdms** – a nova.objects.BlockDeviceMappingList for the instance
- **block_device_obj** – A nova.objects.BlockDeviceMapping instance with all info about the requested block device. device_name does not need to be set, and should be decided by the driver implementation if not set.

Returns The chosen device name.

get_diagnostics(*instance*)
Return data about VM diagnostics.

Parameters **instance** – nova.objects.instance.Instance

get_host_cpu_stats()
Get the currently known host CPU stats.

Returns

a dict containing the CPU stat info, eg:

{'kernel': kern,
'idle': idle,
'user': user,
'iowait': wait,
'frequency': freq},

where kern and user indicate the cumulative CPU time (nanoseconds) spent by kernel and user processes respectively, idle indicates the cumulative idle CPU time (nanoseconds), wait indicates the cumulative I/O wait CPU time (nanoseconds), since the host is booting up; freq indicates the current CPU frequency (MHz). All values are long integers.

get_host_ip_addr()
Retrieves the IP address of the dom0

get_host_uptime()
Returns the result of calling "uptime" on the target host.

get_info(*instance*)
Get the current status of an instance, by name (not ID!)

Parameters **instance** – nova.objects.instance.Instance object

Returns a InstanceInfo object

get_instance_diagnostics(*instance*)
Return data about VM diagnostics.

Parameters **instance** – nova.objects.instance.Instance

get_instance_disk_info(*instance*, *block_device_info=None*)
Retrieve information about actual disk sizes of an instance.

Parameters

- **instance** – nova.objects.Instance
- **block_device_info** – Optional; Can be used to filter out devices which are actually volumes.

Returns

json strings with below format:

```
"[{'path':'disk',
   'type':'raw',
   'virt_disk_size':'10737418240',
   'backing_file':'backing_file',
   'disk_size':'83886080'
   'over_committed_disk_size':'10737418240'},
   ...]"
```

get_num_instances()

Return the total number of virtual machines.

Return the number of virtual machines that the hypervisor knows about.

Note: This implementation works for all drivers, but it is not particularly efficient. Maintainers of the virt drivers are encouraged to override this method with something more efficient.

---

get_per_instance_usage()

Get information about instance resource usage.

Returns dict of nova uuid => dict of usage info

get_rdp_console(*context*, *instance*)

Get connection info for a rdp console.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance

:returns an instance of console.type.ConsoleRDP

get_serial_console(*context*, *instance*)

Get connection info for a serial console.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance

:returns an instance of console.type.ConsoleSerial

get_spice_console(*context*, *instance*)

Get connection info for a spice console.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance

:returns an instance of console.type.ConsoleSpice

get_vnc_console(*context*, *instance*)

Get connection info for a vnc console.

Parameters

- **context** – security context

- **instance** – nova.objects.instance.Instance

:returns an instance of console.type.ConsoleVNC

get_volume_connector(*instance*)
Get connector information for the instance for attaching to volumes.

Connector information is a dictionary representing the ip of the machine that will be making the connection, the name of the iscsi initiator and the hostname of the machine as follows:

```
{
    'ip': ip,
    'initiator': initiator,
    'host': hostname
}
```

host_maintenance_mode(*host*, *mode*)
Start/Stop host maintenance window. On start, it triggers guest VMs evacuation.

host_power_action(*action*)
Reboots, shuts down or powers up the host.

init_host(*host*)
Initialize anything that is necessary for the driver to function, including catching up with currently running VM's on the given host.

inject_file(*instance*, *b64_path*, *b64_contents*)
Writes a file on the specified instance.

The first parameter is an instance of nova.compute.service.Instance, and so the instance is being specified as instance.name. The second parameter is the base64-encoded path to which the file is to be written on the instance; the third is the contents of the file, also base64-encoded.

NOTE(russellb) This method is deprecated and will be removed once it can be removed from nova.compute.manager.

inject_network_info(*instance*, *nw_info*)
inject network info for specified instance.

instance_exists(*instance*)
Checks existence of an instance on the host.

Parameters **instance** – The instance to lookup

Returns True if an instance with the supplied ID exists on the host, False otherwise.

Note: This implementation works for all drivers, but it is not particularly efficient. Maintainers of the virt drivers are encouraged to override this method with something more efficient.

---

instance_on_disk(*instance*)
Checks access of instance files on the host.

Parameters **instance** – nova.objects.instance.Instance to lookup

Returns True if files of an instance with the supplied ID accessible on the host, False otherwise.

Note: Used in rebuild for HA implementation and required for validation of access to instance shared disk files

---

is_supported_fs_format(*fs_type*)
Check whether the file format is supported by this driver

Parameters **fs_type** – the file system type to be checked, the validate values are defined at disk API module.

list_instance_uuids()
Return the UUIDS of all the instances known to the virtualization layer, as a list.

list_instances()
Return the names of all the instances known to the virtualization layer, as a list.

live_migration(*context*, *instance*, *dest*, *post_method*, *recover_method*, *block_migration=False*, *migrate_data=None*)
Live migration of an instance to another host.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance object instance object that is migrated.
- **dest** – destination host
- **post_method** – post operation method. expected nova.compute.manager._post_live_migration.
- **recover_method** – recovery method when any exception occurs. expected nova.compute.manager._rollback_live_migration.
- **block_migration** – if true, migrate VM disk.
- **migrate_data** – implementation specific params.

macs_for_instance(*instance*)
What MAC addresses must this instance have?

Some hypervisors (such as bare metal) cannot do freeform virtualisation of MAC addresses. This method allows drivers to return a set of MAC addresses that the instance is to have. allocate_for_instance will take this into consideration when provisioning networking for the instance.

Mapping of MAC addresses to actual networks (or permitting them to be freeform) is up to the network implementation layer. For instance, with openflow switches, fixed MAC addresses can still be virtualised onto any L2 domain, with arbitrary VLANs etc, but regular switches require pre-configured MAC->network mappings that will match the actual configuration.

Most hypervisors can use the default implementation which returns None. Hypervisors with MAC limits should return a set of MAC addresses, which will be supplied to the allocate_for_instance call by the compute manager, and it is up to that call to ensure that all assigned network details are compatible with the set of MAC addresses.

This is called during spawn_instance by the compute manager.

Returns None, or a set of MAC ids (e.g. set(['12:34:56:78:90:ab'])). None means 'no constraints', a set means 'these and only these MAC addresses'.

manage_image_cache(*context*, *all_instances*)
Manage the driver's local image cache.

Some drivers chose to cache images for instances on disk. This method is an opportunity to do management of that cache which isn't directly related to other calls into the driver. The prime example is to clean the cache and remove images which are no longer of interest.

Parameters **all_instances** – nova.objects.instance.InstanceList

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)
Transfers the disk of a running instance in multiple phases, turning off the instance before the end.

Parameters

- **instance** – nova.objects.instance.Instance
- **timeout** – time to wait for GuestOS to shutdown
- **retry_interval** – How often to signal guest while waiting for it to shutdown

need_legacy_block_device_info
Tell the caller if the driver requires legacy block device info.

Tell the caller whether we expect the legacy format of block device info to be passed in to methods that expect it.

node_is_available(*nodename*)
Return whether this compute service manages a particular node.

pause(*instance*)
Pause the specified instance.

Parameters **instance** – nova.objects.instance.Instance

plug_vifs(*instance*, *network_info*)
Plug VIFs into networks.

Parameters **instance** – nova.objects.instance.Instance

poll_rebooting_instances(*timeout*, *instances*)
Poll for rebooting instances

Parameters

- **timeout** – the currently configured timeout for considering rebooting instances to be stuck
- **instances** – instances that have been in rebooting state longer than the configured timeout

post_interrupted_snapshot_cleanup(*context*, *instance*)
Cleans up any resources left after an interrupted snapshot.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance

post_live_migration(*context*, *instance*, *block_device_info*, *migrate_data=None*)
Post operation of live migration at source host.

Parameters

- **context** – security context
- **migrate_data** – if not None, it is a dict which has data

Instance instance object that was migrated

Block_device_info instance block device information

post_live_migration_at_destination(*context*, *instance*, *network_info*, *block_migration=False*, *block_device_info=None*)
Post operation of live migration at destination host.

Parameters

- **context** – security context
- **instance** – instance object that is migrated
- **network_info** – instance network information
- **block_migration** – if true, post operation of block_migration.

post_live_migration_at_source(*context*, *instance*, *network_info*)
Unplug VIFs from networks at source.

Parameters

- **context** – security context
- **instance** – instance object reference
- **network_info** – instance network information

power_off(*instance*, *timeout=0*, *retry_interval=0*)
Power off the specified instance.

Parameters

- **instance** – nova.objects.instance.Instance
- **timeout** – time to wait for GuestOS to shutdown
- **retry_interval** – How often to signal guest while waiting for it to shutdown

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)
Power on the specified instance.

Parameters **instance** – nova.objects.instance.Instance

pre_live_migration(*context*, *instance*, *block_device_info*, *network_info*, *disk_info*, *migrate_data=None*)
Prepare an instance for live migration

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance object
- **block_device_info** – instance block device information
- **network_info** – instance network information
- **disk_info** – instance disk information
- **migrate_data** – implementation specific data dict.

quiesce(*context*, *instance*, *image_meta*)
Quiesce the specified instance to prepare for snapshots.

If the specified instance doesn't support quiescing, InstanceQuiesceNotSupported is raised. When it fails to quiesce by other errors (e.g. agent timeout), NovaException is raised.

Parameters

- **context** – request context
- **instance** – nova.objects.instance.Instance to be quiesced
- **image_meta** – image object returned by nova.image.glance that defines the image from which this instance was created

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)
Reboot the specified instance.

After this is called successfully, the instance's state goes back to power_state.RUNNING. The virtualization platform should ensure that the reboot action has completed successfully even in cases in which the underlying domain/vm is paused or halted/stopped.

Parameters

- **instance** – nova.objects.instance.Instance
- **network_info** – `get_instance_nw_info()`
- **reboot_type** – Either a HARD or SOFT reboot
- **block_device_info** – Info pertaining to attached volumes
- **bad_volumes_callback** – Function to handle any bad volumes encountered

rebuild(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *bdms*, *detach_block_devices*, *attach_block_devices*, *network_info=None*, *recreate=False*, *block_device_info=None*, *preserve_ephemeral=False*)
Destroy and re-make this instance.

A 'rebuild' effectively purges all existing data from the system and remakes the VM with given 'metadata' and 'personalities'.

This base class method shuts down the VM, detaches all block devices, then spins up the new VM afterwards. It may be overridden by hypervisors that need to - e.g. for optimisations, or when the 'VM' is actually proxied and needs to be held across the shutdown + spin up steps.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance This function should use the data there to guide the creation of the new instance.
- **image_meta** – image object returned by nova.image.glance that defines the image from which to boot this instance
- **injected_files** – User files to inject into instance.
- **admin_password** – Administrator password to set in instance.
- **bdms** – block-device-mappings to use for rebuild
- **detach_block_devices** – function to detach block devices. See nova.compute.manager.ComputeManager:_rebuild_default_impl for usage.
- **attach_block_devices** – function to attach block devices. See nova.compute.manager.ComputeManager:_rebuild_default_impl for usage.
- **network_info** – `get_instance_nw_info()`
- **recreate** – True if the instance is being recreated on a new hypervisor - all the cleanup of old state is skipped.
- **block_device_info** – Information about block devices to be attached to the instance.
- **preserve_ephemeral** – True if the default ephemeral storage partition must be preserved on rebuild

refresh_instance_security_rules(*instance*)
Refresh security group rules

Gets called when an instance gets added to or removed from the security group the instance is a member of or if the group gains or loses a rule.

refresh_provider_fw_rules()
This triggers a firewall update based on database changes.

When this is called, rules have either been added or removed from the datastore. You can retrieve rules with `nova.db.provider_fw_rule_get_all()`.

Provider rules take precedence over security group rules. If an IP would be allowed by a security group ingress rule, but blocked by a provider rule, then packets from the IP are dropped. This includes intra-project traffic in the case of the allow_project_net_traffic flag for the libvirt-derived classes.

refresh_security_group_members(*security_group_id*)
This method is called when a security group is added to an instance.

This message is sent to the virtualization drivers on hosts that are running an instance that belongs to a security group that has a rule that references the security group identified by *security_group_id*. It is the responsibility of this method to make sure any rules that authorize traffic flow with members of the security group are updated and any new members can communicate, and any removed members cannot.

Scenario:

- we are running on host 'H0' and we have an instance 'i-0'.
- instance 'i-0' is a member of security group 'speaks-b'
- group 'speaks-b' has an ingress rule that authorizes group 'b'
- another host 'H1' runs an instance 'i-1'
- instance 'i-1' is a member of security group 'b'

When 'i-1' launches or terminates we will receive the message to update members of group 'b', at which time we will make any changes needed to the rules for instance 'i-0' to allow or deny traffic coming from 'i-1', depending on if it is being added or removed from the group.

In this scenario, 'i-1' could just as easily have been running on our host 'H0' and this method would still have been called. The point was that this method isn't called on the host where instances of that group are running (as is the case with `refresh_security_group_rules()`) but is called where references are made to authorizing those instances.

An error should be raised if the operation cannot complete.

refresh_security_group_rules(*security_group_id*)
This method is called after a change to security groups.

All security groups and their associated rules live in the datastore, and calling this method should apply the updated rules to instances running the specified security group.

An error should be raised if the operation cannot complete.

register_event_listener(*callback*)
Register a callback to receive events.

Register a callback to receive asynchronous event notifications from hypervisors. The callback will be invoked with a single parameter, which will be an instance of the nova.virt.event.Event class.

remove_from_aggregate(*context*, *aggregate*, *host*, ***kwargs*)
Remove a compute host from an aggregate.

rescue(*context*, *instance*, *network_info*, *image_meta*, *rescue_password*)
Rescue the specified instance.

Parameters **instance** – nova.objects.instance.Instance

reset_network(*instance*)
reset networking for specified instance.

restore(*instance*)
Restore the specified instance.

Parameters **instance** – nova.objects.instance.Instance

resume(*context*, *instance*, *network_info*, *block_device_info=None*)
resume the specified instance.

Parameters

- **context** – the context for the resume
- **instance** – nova.objects.instance.Instance being resumed
- **network_info** – `get_instance_nw_info()`
- **block_device_info** – instance volume block device info

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)
resume guest state when a host is booted.

Parameters **instance** – nova.objects.instance.Instance

rollback_live_migration_at_destination(*context*, *instance*, *network_info*, *block_device_info*, *destroy_disks=True*, *migrate_data=None*)
Clean up destination node after a failed live migration.

Parameters

- **context** – security context
- **instance** – instance object that was being migrated
- **network_info** – instance network information
- **block_device_info** – instance block device information
- **destroy_disks** – if true, destroy disks at destination during cleanup
- **migrate_data** – implementation specific params

set_admin_password(*instance*, *new_pass*)
Set the root password on the specified instance.

Parameters

- **instance** – nova.objects.instance.Instance
- **new_pass** – the new password

set_bootable(*instance*, *is_bootable*)
Set the ability to power on/off an instance.

Parameters **instance** – nova.objects.instance.Instance

set_host_enabled(*enabled*)
Sets the specified host's ability to accept new instances.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Snapshots the specified instance.

Parameters

- **context** – security context

- **instance** – nova.objects.instance.Instance
- **image_id** – Reference to a pre-created image that will hold the snapshot.

soft_delete(*instance*)
Soft delete the specified instance.

Parameters **instance** – nova.objects.instance.Instance

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)
Create a new instance/VM/domain on the virtualization platform.

Once this successfully completes, the instance should be running (power_state.RUNNING).

If this fails, any partial instance should be completely cleaned up, and the virtualization platform should be in the state that it was before this call began.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance This function should use the data there to guide the creation of the new instance.
- **image_meta** – image object returned by nova.image.glance that defines the image from which to boot this instance
- **injected_files** – User files to inject into instance.
- **admin_password** – Administrator password to set in instance.
- **network_info** – `get_instance_nw_info()`
- **block_device_info** – Information about block devices to be attached to the instance.

suspend(*context*, *instance*)
suspend the specified instance.

Parameters

- **context** – the context for the suspend
- **instance** – nova.objects.instance.Instance

swap_volume(*old_connection_info*, *new_connection_info*, *instance*, *mountpoint*, *resize_to*)
Replace the disk attached to the instance.

Parameters

- **instance** – nova.objects.instance.Instance
- **resize_to** – This parameter is used to indicate the new volume size when the new volume lager than old volume. And the units is Gigabyte.

undo_aggregate_operation(*context*, *op*, *aggregate*, *host*, *set_error=True*)
Undo for Resource Pools.

unfilter_instance(*instance*, *network_info*)
Stop filtering instance.

unpause(*instance*)
Unpause paused VM instance.

Parameters **instance** – nova.objects.instance.Instance

unplug_vifs(*instance*, *network_info*)
Unplug VIFs from networks.

Parameters **instance** – nova.objects.instance.Instance

unquiesce(*context*, *instance*, *image_meta*)
Unquiesce the specified instance after snapshots.

If the specified instance doesn't support quiescing, InstanceQuiesceNotSupported is raised. When it fails to quiesce by other errors (e.g. agent timeout), NovaException is raised.

Parameters

- **context** – request context
- **instance** – nova.objects.instance.Instance to be unquiesced
- **image_meta** – image object returned by nova.image.glance that defines the image from which this instance was created

unrescue(*instance*, *network_info*)
Unrescue the specified instance.

Parameters **instance** – nova.objects.instance.Instance

volume_snapshot_create(*context*, *instance*, *volume_id*, *create_info*)
Snapshots volumes attached to a specified instance.

Parameters

- **context** – request context
- **instance** – nova.objects.instance.Instance that has the volume attached
- **volume_id** – Volume to be snapshotted
- **create_info** – The data needed for nova to be able to attach to the volume. This is the same data format returned by Cinder's initialize_connection() API call. In the case of doing a snapshot, it is the image file Cinder expects to be used as the active disk after the snapshot operation has completed. There may be other data included as well that is needed for creating the snapshot.

volume_snapshot_delete(*context*, *instance*, *volume_id*, *snapshot_id*, *delete_info*)
Snapshots volumes attached to a specified instance.

Parameters

- **context** – request context
- **instance** – nova.objects.instance.Instance that has the volume attached
- **volume_id** – Attached volume associated with the snapshot
- **snapshot_id** – The snapshot to delete.
- **delete_info** – Volume backend technology specific data needed to be able to complete the snapshot. For example, in the case of qcow2 backed snapshots, this would include the file being merged, and the file being merged into (if appropriate).

block_device_info_get_ephemerals(*block_device_info*)

block_device_info_get_mapping(*block_device_info*)

block_device_info_get_root(*block_device_info*)

block_device_info_get_swap(*block_device_info*)

compute_driver_matches(*match*)

driver_dict_from_config(*named_driver_config*, **args*, ***kwargs*)

get_block_device_info(*instance*, *block_device_mapping*)
Converts block device mappings for an instance to driver format.

Virt drivers expect block device mapping to be presented in the format of a dict containing the following keys:

- root_device_name: device name of the root disk
- **ephemerals: a (potentially empty) list of DriverEphemeralBlockDevice** instances
- swap: An instance of DriverSwapBlockDevice or None
- **block_device_mapping: a (potentially empty) list of** DriverVolumeBlockDevice or any of it's more specialized subclasses.

load_compute_driver(*virtapi*, *compute_driver=None*)
Load a compute driver module.

Load the compute driver module specified by the compute_driver configuration option or, if supplied, the driver name supplied as an argument.

Compute drivers constructors take a VirtAPI object as their first object and this must be supplied.

Parameters

- **virtapi** – a VirtAPI instance
- **compute_driver** – a compute driver name to override the config opt

Returns a ComputeDriver instance

swap_is_usable(*swap*)

3.7.639 The nova.virt.event Module

Asynchronous event notifications from virtualization drivers.

This module defines a set of classes representing data for various asynchronous events that can occur in a virtualization driver.

class Event(*timestamp=None*)
Bases: object

Base class for all events emitted by a hypervisor.

All events emitted by a virtualization driver are subclasses of this base object. The only generic information recorded in the base class is a timestamp indicating when the event first occurred. The timestamp is recorded as fractional seconds since the UNIX epoch.

get_timestamp()

class InstanceEvent(*uuid*, *timestamp=None*)
Bases: nova.virt.event.Event

Base class for all instance events.

All events emitted by a virtualization driver which are associated with a virtual domain instance are subclasses of this base object. This object records the UUID associated with the instance.

get_instance_uuid()

class LifecycleEvent(*uuid*, *transition*, *timestamp=None*)
Bases: nova.virt.event.InstanceEvent

Class for instance lifecycle state change events.

When a virtual domain instance lifecycle state changes, events of this class are emitted. The EVENT_LIFECYCLE_XX constants defined why lifecycle change occurred. This event allows detection of an instance starting/stopping without need for polling.

get_name()

get_transition()

3.7.640 The nova.virt.fake Module

A fake (in-memory) hypervisor+api.

Allows nova testing w/o a hypervisor. This module also documents the semantics of real hypervisor connections.

class FakeDriver(*virtapi*, *read_only=False*)

Bases: nova.virt.driver.ComputeDriver

attach_interface(*instance*, *image_meta*, *vif*)

attach_volume(*context*, *connection_info*, *instance*, *mountpoint*, *disk_bus=None*, *device_type=None*, *encryption=None*)

Attach the disk to the instance at mountpoint using info.

block_stats(*instance*, *disk_id*)

capabilities = {'supports_recreate': True, 'has_imagecache': True, 'supports_migrate_to_same_host': True}

check_can_live_migrate_destination(*context*, *instance*, *src_compute_info*, *dst_compute_info*, *block_migration=False*, *disk_over_commit=False*)

check_can_live_migrate_destination_cleanup(*context*, *dest_check_data*)

check_can_live_migrate_source(*context*, *instance*, *dest_check_data*, *block_device_info=None*)

cleanup(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*, *destroy_vifs=True*)

confirm_migration(*migration*, *instance*, *network_info*)

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)

detach_interface(*instance*, *vif*)

detach_volume(*connection_info*, *instance*, *mountpoint*, *encryption=None*)

Detach the disk attached to the instance.

ensure_filtering_rules_for_instance(*instance*, *network_info*)

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)

get_all_bw_counters(*instances*)

Return bandwidth usage counters for each interface on each running VM.

get_all_volume_usage(*context*, *compute_host_bdms*)

Return usage info for volumes attached to vms on a given host.

get_available_nodes(*refresh=False*)

get_available_resource(*nodename*)
Updates compute manager resource info on ComputeNode table.

Since we don't have a real hypervisor, pretend we have lots of disk and ram.

get_console_output(*context*, *instance*)

get_console_pool_info(*console_type*)

get_diagnostics(*instance*)

get_host_cpu_stats()

get_host_ip_addr()

get_info(*instance*)

get_instance_diagnostics(*instance*)

get_instance_disk_info(*instance*, *block_device_info=None*)

get_rdp_console(*context*, *instance*)

get_serial_console(*context*, *instance*)

get_spice_console(*context*, *instance*)

get_vnc_console(*context*, *instance*)

get_volume_connector(*instance*)

host_maintenance_mode(*host*, *mode*)
Start/Stop host maintenance window. On start, it triggers guest VMs evacuation.

host_power_action(*action*)
Reboots, shuts down or powers up the host.

init_host(*host*)

inject_file(*instance*, *b64_path*, *b64_contents*)

instance_on_disk(*instance*)

list_instance_uuids()

list_instances()

live_migration(*context*, *instance*, *dest*, *post_method*, *recover_method*, *block_migration=False*, *migrate_data=None*)

local_gb = 600000
Fake hypervisor driver.

memory_mb = 800000

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)

pause(*instance*)

plug_vifs(*instance*, *network_info*)
Plug VIFs into networks.

poll_rebooting_instances(*timeout*, *instances*)

post_live_migration_at_destination(*context*, *instance*, *network_info*, *block_migration=False*, *block_device_info=None*)

power_off(*instance*, *timeout=0*, *retry_interval=0*)

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)

pre_live_migration(*context*, *instance*, *block_device_info*, *network_info*, *disk_info*, *migrate_data=None*)

quiesce(*context*, *instance*, *image_meta*)

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)

refresh_instance_security_rules(*instance*)

refresh_provider_fw_rules()

refresh_security_group_members(*security_group_id*)

refresh_security_group_rules(*security_group_id*)

rescue(*context*, *instance*, *network_info*, *image_meta*, *rescue_password*)

restore(*instance*)

resume(*context*, *instance*, *network_info*, *block_device_info=None*)

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)

set_admin_password(*instance*, *new_pass*)

set_host_enabled(*enabled*)
Sets the specified host's ability to accept new instances.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)

soft_delete(*instance*)

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)

suspend(*context*, *instance*)

swap_volume(*old_connection_info*, *new_connection_info*, *instance*, *mountpoint*, *resize_to*)
Replace the disk attached to the instance.

unfilter_instance(*instance*, *network_info*)

unpause(*instance*)

unplug_vifs(*instance*, *network_info*)
Unplug VIFs from networks.

unquiesce(*context*, *instance*, *image_meta*)

unrescue(*instance*, *network_info*)

vcpus = 1000

class **FakeInstance**(*name*, *state*, *uuid*)
Bases: `object`

class **FakeVirtAPI**
Bases: `nova.virt.virtapi.VirtAPI`

provider_fw_rule_get_all(*context*)

wait_for_instance_event(**args*, ***kwds*)

class **Resources**(*vcpus=8*, *memory_mb=8000*, *local_gb=500*)
Bases: `object`

claim(*vcpus=0*, *mem=0*, *disk=0*)

dump()

local_gb = 0

local_gb_used = 0

memory_mb = 0

memory_mb_used = 0

release(*vcpus=0*, *mem=0*, *disk=0*)

vcpus = 0

vcpus_used = 0

class SmallFakeDriver(*virtapi*, *read_only=False*)
Bases: nova.virt.fake.FakeDriver

local_gb = 1028

memory_mb = 8192

vcpus = 1

restore_nodes()
Resets FakeDriver's node list modified by set_nodes().

Usually called from tearDown().

set_nodes(*nodes*)
Sets FakeDriver's node.list.

It has effect on the following methods: get_available_nodes() get_available_resource

To restore the change, call restore_nodes()

3.7.641 The nova.virt.firewall Module

class FirewallDriver(*virtapi*)
Bases: object

Firewall Driver base class.

Defines methods that any driver providing security groups and provider firewall functionality should implement.

apply_instance_filter(*instance*, *network_info*)
Apply instance filter.

Once this method returns, the instance should be firewalled appropriately. This method should as far as possible be a no-op. It's vastly preferred to get everything set up in prepare_instance_filter.

filter_defer_apply_off()
Turn off deferral of IPTables rules and apply the rules now.

filter_defer_apply_on()
Defer application of IPTables rules.

instance_filter_exists(*instance*, *network_info*)
Check nova-instance-instance-xxx exists.

prepare_instance_filter(*instance*, *network_info*)
Prepare filters for the instance.

At this point, the instance isn't running yet.

refresh_instance_security_rules(*instance*)
Refresh security group rules from data store

Gets called when an instance gets added to or removed from the security group the instance is a member of or if the group gains or looses a rule.

refresh_provider_fw_rules()
Refresh common rules for all hosts/instances from data store.

Gets called when a rule has been added to or removed from the list of rules (via admin api).

refresh_security_group_members(*security_group_id*)
Refresh security group members from data store

Gets called when an instance gets added to or removed from the security group.

refresh_security_group_rules(*security_group_id*)
Refresh security group rules from data store

Gets called when a rule has been added to or removed from the security group.

setup_basic_filtering(*instance*, *network_info*)
Create rules to block spoofing and allow dhcp.

This gets called when spawning an instance, before `prepare_instance_filter()`.

unfilter_instance(*instance*, *network_info*)
Stop filtering instance.

class **IptablesFirewallDriver**(*virtapi*, ***kwargs*)
Bases: `nova.virt.firewall.FirewallDriver`

Driver which enforces security groups through iptables rules.

add_filters_for_instance(*instance*, *network_info*, *inst_ipv4_rules*, *inst_ipv6_rules*)

apply_instance_filter(*instance*, *network_info*)
No-op. Everything is done in prepare_instance_filter.

do_refresh_instance_rules(*instance*)

do_refresh_security_group_rules(*security_group*)

filter_defer_apply_off()

filter_defer_apply_on()

instance_filter_exists(*instance*, *network_info*)

instance_rules(*instance*, *network_info*)

prepare_instance_filter(*instance*, *network_info*)

refresh_instance_security_rules(*instance*)

refresh_provider_fw_rules()
See `FirewallDriver` docs.

refresh_security_group_members(*security_group*)

refresh_security_group_rules(*security_group*)

remove_filters_for_instance(*instance*)

setup_basic_filtering(*instance*, *network_info*)

unfilter_instance(*instance*, *network_info*)

class NoopFirewallDriver(**args*, ***kwargs*)
Bases: object

Firewall driver which just provides No-op methods.

instance_filter_exists(*instance*, *network_info*)

load_driver(*default*, **args*, ***kwargs*)

3.7.642 The nova.virt.hardware Module

class InstanceInfo(*state=None*, *max_mem_kb=0*, *mem_kb=0*, *num_cpu=0*, *cpu_time_ns=0*, *id=None*)
Bases: object

format_cpu_spec(*cpuset*, *allow_ranges=True*)
Format a libvirt CPU range specification.

Parameters **cpuset** – set (or list) of CPU indexes

Format a set/list of CPU indexes as a libvirt CPU range specification. It allow_ranges is true, it will try to detect continuous ranges of CPUs, otherwise it will just list each CPU index explicitly.

Returns a formatted CPU range string

get_best_cpu_topology(*flavor*, *image_meta*, *allow_threads=True*, *numa_topology=None*)
Get best CPU topology according to settings

Parameters

- **flavor** – Flavor object to query extra specs from
- **image_meta** – ImageMeta object to query properties from
- **allow_threads** – if the hypervisor supports CPU threads
- **numa_topology** – InstanceNUMATopology object that may contain additional topology constraints (such as threading information) that we should consider

Look at the properties set in the flavor extra specs and the image metadata and build up a list of all possible valid CPU topologies that can be used in the guest. Then return the best topology to use

Returns a nova.objects.VirtCPUTopology instance for best topology

get_host_numa_usage_from_instance(*host*, *instance*, *free=False*, *never_serialize_result=False*)
Calculate new 'numa_usage' of 'host' from 'instance' NUMA usage

This is a convenience method to help us handle the fact that we use several different types throughout the code (ComputeNode and Instance objects, dicts, scheduler HostState) which may have both json and deserialized versions of objects.numa classes.

Handles all the complexity without polluting the class method with it.

Parameters

- **host** – nova.objects.ComputeNode instance, or a db object or dict
- **instance** – nova.objects.Instance instance, or a db object or dict
- **free** – if True the the returned topology will have it's usage decreased instead.
- **never_serialize_result** – if True result will always be an instance of objects.NUMATopology class.

Returns numa_usage in the format it was on the host or objects.NUMATopology instance if never_serialize_result was True

get_number_of_serial_ports(*flavor*, *image_meta*)
Get the number of serial consoles from the flavor or image

Parameters

- **flavor** – Flavor object to read extra specs from
- **image_meta** – Image object to read image metadata from

If flavor extra specs is not set, then any image meta value is permitted. If flavour extra specs *is* set, then this provides the default serial port count. The image meta is permitted to override the extra specs, but *only* with a lower value. ie

•flavor hw:serial_port_count=4 VM gets 4 serial ports

•flavor hw:serial_port_count=4 and image hw_serial_port_count=2 VM gets 2 serial ports

•image hw_serial_port_count=6 VM gets 6 serial ports

•flavor hw:serial_port_count=4 and image hw_serial_port_count=6 Abort guest boot - forbidden to exceed flavor value

Returns number of serial ports

get_vcpu_pin_set()
Parsing vcpu_pin_set config.

Returns a set of pcpu ids can be used by instances.

host_topology_and_format_from_host(*host*)
Convenience method for getting the numa_topology out of hosts

Since we may get a host as either a dict, a db object, or an actual ComputeNode object, or an instance of HostState class, this makes sure we get beck either None, or an instance of objects.NUMATopology class.

Returns A two-tuple, first element is the topology itself or None, second is a boolean set to True if topology was in json format.

instance_topology_from_instance(*instance*)
Convenience method for getting the numa_topology out of instances

Since we may get an Instance as either a dict, a db object, or an actual Instance object, this makes sure we get beck either None, or an instance of objects.InstanceNUMATopology class.

numa_fit_instance_to_host(*host_topology*, *instance_topology*, *limits=None*, *pci_requests=None*, *pci_stats=None*)
Fit the instance topology onto the host topology given the limits

Parameters

- **host_topology** – objects.NUMATopology object to fit an instance on
- **instance_topology** – objects.InstanceNUMATopology to be fitted
- **limits** – objects.NUMATopologyLimits that defines limits
- **pci_requests** – instance pci_requests
- **pci_stats** – pci_stats for the host

Given a host and instance topology and optionally limits - this method will attempt to fit instance cells onto all permutations of host cells by calling the _numa_fit_instance_cell method, and return a new InstanceNUMATopology with it's cell ids set to host cell id's of the first successful permutation, or None.

numa_get_constraints(*flavor*, *image_meta*)
Return topology related to input request

Parameters

- **flavor** – Flavor object to read extra specs from
- **image_meta** – Image object to read image metadata from

Returns InstanceNUMATopology or None

numa_usage_from_instances(*host*, *instances*, *free=False*)
Get host topology usage

Parameters

- **host** – objects.NUMATopology with usage information
- **instances** – list of objects.InstanceNUMATopology
- **free** – If True usage of the host will be decreased

Sum the usage from all @instances to report the overall host topology usage

Returns objects.NUMATopology including usage information

parse_cpu_spec(*spec*)
Parse a CPU set specification.

Parameters **spec** – cpu set string eg "1-4,^3,6"

Each element in the list is either a single CPU number, a range of CPU numbers, or a caret followed by a CPU number to be excluded from a previous range.

Returns a set of CPU indexes

3.7.643 The nova.virt.hyperv.basevolumeutils Module

Helper methods for operations related to the management of volumes, and storage repositories

class BaseVolumeUtils(*host='.'*)
Bases: object

execute_log_out(*session_id*)

get_device_number_for_target(*target_iqn*, *target_lun*)

get_iscsi_initiator()
Get iscsi initiator name for this machine.

get_session_id_from_mounted_disk(*physical_drive_path*)

get_target_from_disk_path(*disk_path*)

get_target_lun_count(*target_iqn*)

login_storage_target(*target_lun*, *target_iqn*, *target_portal*)

logout_storage_target(*target_iqn*)

volume_in_mapping(*mount_device*, *block_device_info*)

3.7.644 The nova.virt.hyperv.constants Module

Constants used in ops classes

3.7.645 The nova.virt.hyperv.driver Module

A Hyper-V Nova Compute driver.

class HyperVDriver(*virtapi*)

Bases: nova.virt.driver.ComputeDriver

attach_volume(*context*, *connection_info*, *instance*, *mountpoint*, *disk_bus=None*, *device_type=None*, *encryption=None*)

capabilities = {'supports_recreate': False, 'has_imagecache': False, 'supports_migrate_to_same_host': True}

check_can_live_migrate_destination(*context*, *instance*, *src_compute_info*, *dst_compute_info*, *block_migration=False*, *disk_over_commit=False*)

check_can_live_migrate_destination_cleanup(*context*, *dest_check_data*)

check_can_live_migrate_source(*context*, *instance*, *dest_check_data*, *block_device_info=None*)

cleanup(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*, *destroy_vifs=True*)

Cleanup after instance being destroyed by Hypervisor.

confirm_migration(*migration*, *instance*, *network_info*)

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)

detach_volume(*connection_info*, *instance*, *mountpoint*, *encryption=None*)

ensure_filtering_rules_for_instance(*instance*, *network_info*)

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)

get_available_nodes(*refresh=False*)

get_available_resource(*nodename*)

get_console_output(*context*, *instance*)

get_host_ip_addr()

get_host_uptime()

get_info(*instance*)

get_instance_disk_info(*instance*, *block_device_info=None*)

get_rdp_console(*context*, *instance*)

get_volume_connector(*instance*)

host_power_action(*action*)

init_host(*host*)

list_instance_uuids()

list_instances()

live_migration(*context*, *instance*, *dest*, *post_method*, *recover_method*, *block_migration=False*, *migrate_data=None*)

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)

pause(*instance*)

plug_vifs(*instance*, *network_info*)
Plug VIFs into networks.

post_live_migration(*context*, *instance*, *block_device_info*, *migrate_data=None*)

post_live_migration_at_destination(*context*, *instance*, *network_info*, *block_migration=False*, *block_device_info=None*)

power_off(*instance*, *timeout=0*, *retry_interval=0*)

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)

pre_live_migration(*context*, *instance*, *block_device_info*, *network_info*, *disk_info*, *migrate_data=None*)

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)

resume(*context*, *instance*, *network_info*, *block_device_info=None*)

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)
Resume guest state when a host is booted.

rollback_live_migration_at_destination(*context*, *instance*, *network_info*, *block_device_info*, *destroy_disks=True*, *migrate_data=None*)

snapshot(*context*, *instance*, *image_id*, *update_task_state*)

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)

suspend(*context*, *instance*)

unfilter_instance(*instance*, *network_info*)

unpause(*instance*)

unplug_vifs(*instance*, *network_info*)
Unplug VIFs from networks.

3.7.646 The nova.virt.hyperv.hostops Module

Management class for host operations.

class HostOps
Bases: object

get_available_resource()
Retrieve resource info.

This method is called when nova-compute launches, and as part of a periodic task.

Returns dictionary describing resources

get_host_ip_addr()

get_host_uptime()
Returns the host uptime.

host_power_action(*action*)
Reboots, shuts down or powers up the host.

3.7.647 The nova.virt.hyperv.hostutils Module

class HostUtils
Bases: object

check_min_windows_version(*major*, *minor*, *build=0*)

get_cpus_info()

get_default_vm_generation()

get_host_tick_count64()

get_local_ips()

get_memory_info()
Returns a tuple with total visible memory and free physical memory expressed in kB.

get_supported_vm_types()
Get the supported Hyper-V VM generations. Hyper-V Generation 2 VMs are supported in Windows 8.1, Windows Server / Hyper-V Server 2012 R2 or newer.

Returns array of supported VM generations (ex. ['hyperv-gen1'])

get_volume_info(*drive*)
Returns a tuple with total size and free space expressed in bytes.

get_windows_version()

host_power_action(*action*)

is_cpu_feature_present(*feature_key*)

3.7.648 The nova.virt.hyperv.hostutilsv2 Module

class HostUtilsV2
Bases: nova.virt.hyperv.hostutils.HostUtils

FEATURE_RDS_VIRTUALIZATION = 322

3.7.649 The nova.virt.hyperv.imagecache Module

Image caching and management.

class ImageCache
Bases: object

get_cached_image(*context*, *instance*)

get_image_details(*context*, *instance*)

3.7.650 The nova.virt.hyperv.ioutils Module

class IOThread(*src*, *dest*, *max_bytes*)
Bases: threading.Thread

is_active()

join()

run()

3.7.651 The nova.virt.hyperv.livemigrationops Module

Management class for live migration VM operations.

class LiveMigrationOps
Bases: object

check_can_live_migrate_destination(**args*, ***kwds*)

check_can_live_migrate_destination_cleanup(**args*, ***kwds*)

check_can_live_migrate_source(**args*, ***kwds*)

live_migration(**args*, ***kwds*)

post_live_migration(**args*, ***kwds*)

post_live_migration_at_destination(**args*, ***kwds*)

pre_live_migration(**args*, ***kwds*)

check_os_version_requirement(*function*)

3.7.652 The nova.virt.hyperv.livemigrationutils Module

class LiveMigrationUtils
Bases: object

check_live_migration_config()

live_migrate_vm(*vm_name*, *dest_host*)

3.7.653 The nova.virt.hyperv.migrationops Module

Management class for migration / resize operations.

class MigrationOps
Bases: object

confirm_migration(*migration*, *instance*, *network_info*)

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance=False*, *block_device_info=None*, *power_on=True*)

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)

3.7.654 The nova.virt.hyperv.networkutils Module

Utility class for network related operations.

class NetworkUtils
Bases: object

create_vswitch_port(*vswitch_path*, *port_name*)

get_external_vswitch(*vswitch_name*)

vswitch_port_needed()

3.7.655 The nova.virt.hyperv.networkutilsv2 Module

Utility class for network related operations. Based on the "root/virtualization/v2" namespace available starting with Hyper-V Server / Windows Server 2012.

class NetworkUtilsV2
Bases: nova.virt.hyperv.networkutils.NetworkUtils

create_vswitch_port(*vswitch_path*, *port_name*)

get_external_vswitch(*vswitch_name*)

vswitch_port_needed()

3.7.656 The nova.virt.hyperv.pathutils Module

class PathUtils
Bases: object

check_smb_mapping(*smbfs_share*)

copy(*src*, *dest*)

copyfile(*src*, *dest*)

exists(*path*)

get_base_vhd_dir()

get_configdrive_path(*instance_name*, *format_ext*, *remote_server=None*)

get_ephemeral_vhd_path(*instance_name*, *format_ext*)

get_export_dir(*instance_name*)

get_instance_dir(*instance_name*, *remote_server=None*, *create_dir=True*, *remove_dir=False*)

get_instance_migr_revert_dir(*instance_name*, *create_dir=False*, *remove_dir=False*)

get_instances_dir(*remote_server=None*)

get_root_vhd_path(*instance_name*, *format_ext*)

get_vm_console_log_paths(*vm_name*, *remote_server=None*)

lookup_configdrive_path(*instance_name*)

lookup_ephemeral_vhd_path(*instance_name*)

lookup_root_vhd_path(*instance_name*)

makedirs(*path*)

mount_smb_share(*smbfs_share*, *username=None*, *password=None*)

move_folder_files(*src_dir*, *dest_dir*)
Moves the files of the given src_dir to dest_dir. It will ignore any nested folders.

Parameters

- **src_dir** – Given folder from which to move files.
- **dest_dir** – Folder to which to move files.

open(*path*, *mode*)
Wrapper on __builtin__.open used to simplify unit testing.

remove(*path*)

rename(*src*, *dest*)

rmtree(*path*)

unmount_smb_share(*smbfs_share*, *force=False*)

3.7.657 The nova.virt.hyperv.rdpconsoleops Module

class RDPConsoleOps
Bases: object

get_rdp_console(*instance*)

3.7.658 The nova.virt.hyperv.rdpconsoleutils Module

class RDPConsoleUtils
Bases: object

get_rdp_console_port()

3.7.659 The nova.virt.hyperv.rdpconsoleutilsv2 Module

class RDPConsoleUtilsV2
Bases: nova.virt.hyperv.rdpconsoleutils.RDPConsoleUtils

get_rdp_console_port()

3.7.660 The nova.virt.hyperv.snapshotops Module

Management class for VM snapshot operations.

class SnapshotOps
Bases: object

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Create snapshot from a running VM instance.

3.7.661 The nova.virt.hyperv.utilsfactory Module

get_hostutils()

get_livemigrationutils()

get_networkutils()

get_pathutils()

get_rdpconsoleutils()

get_vhdutils()

get_vmutils(*host='.'*)

get_volumeutils()

3.7.662 The nova.virt.hyperv.vhdutils Module

Utility class for VHD related operations.

Official VHD format specs can be retrieved at: http://technet.microsoft.com/en-us/library/bb676673.aspx See "Download the Specifications Without Registering"

Official VHDX format specs can be retrieved at: http://www.microsoft.com/en-us/download/details.aspx?id=34750

class VHDUtils

Bases: object

create_differencing_vhd(*path*, *parent_path*)

create_dynamic_vhd(*path*, *max_internal_size*, *format*)

get_best_supported_vhd_format()

get_internal_vhd_size_by_file_size(*vhd_path*, *new_vhd_file_size*)

Fixed VHD size = Data Block size + 512 bytes | Dynamic_VHD_size = Dynamic Disk Header | + Copy of hard disk footer | + Hard Disk Footer | + Data Block | + BAT | Dynamic Disk header fields | Copy of hard disk footer (512 bytes) | Dynamic Disk Header (1024 bytes) | BAT (Block Allocation table) | Data Block 1 | Data Block 2 | Data Block n | Hard Disk Footer (512 bytes) | Default block size is 2M | BAT entry size is 4byte

get_vhd_format(*path*)

get_vhd_info(*vhd_path*)

get_vhd_parent_path(*vhd_path*)

merge_vhd(*src_vhd_path*, *dest_vhd_path*)

reconnect_parent_vhd(*child_vhd_path*, *parent_vhd_path*)

resize_vhd(*vhd_path*, *new_max_size*, *is_file_max_size=True*)

validate_vhd(*vhd_path*)

3.7.663 The nova.virt.hyperv.vhdutilsv2 Module

Utility class for VHD related operations. Based on the "root/virtualization/v2" namespace available starting with Hyper-V Server / Windows Server 2012.

class VHDUtilsV2

Bases: nova.virt.hyperv.vhdutils.VHDUtils

create_differencing_vhd(*path*, *parent_path*)

create_dynamic_vhd(*path*, *max_internal_size*, *format*)

get_best_supported_vhd_format()

get_internal_vhd_size_by_file_size(*vhd_path*, *new_vhd_file_size*)

Get internal size of a VHD according to new VHD file size.

VHDX Size = Header (1MB) + Log + Metadata Region + BAT + Payload Blocks

The chunk size is the maximum number of bytes described by a SB block.

Chunk size = 2^{23} * LogicalSectorSize

Parameters

- **vhd_path** (*str*) – VHD file path
- **new_vhd_file_size** – Size of the new VHD file.

Returns Internal VHD size according to new VHD file size.

get_vhd_info(*vhd_path*)

reconnect_parent_vhd(*child_vhd_path*, *parent_vhd_path*)

3.7.664 The nova.virt.hyperv.vif Module

class HyperVBaseVIFDriver

Bases: object

plug(*instance*, *vif*)

unplug(*instance*, *vif*)

class HyperVNeutronVIFDriver

Bases: nova.virt.hyperv.vif.HyperVBaseVIFDriver

Neutron VIF driver.

plug(*instance*, *vif*)

unplug(*instance*, *vif*)

class HyperVNovaNetworkVIFDriver

Bases: nova.virt.hyperv.vif.HyperVBaseVIFDriver

Nova network VIF driver.

plug(*instance*, *vif*)

unplug(*instance*, *vif*)

3.7.665 The nova.virt.hyperv.vmops Module

Management class for basic VM operations.

class VMOps

Bases: object

attach_config_drive(*instance*, *configdrive_path*, *vm_gen*)

copy_vm_console_logs(*vm_name*, *dest_host*)

copy_vm_dvd_disks(*vm_name*, *dest_host*)

create_ephemeral_vhd(*instance*)

create_instance(*instance*, *network_info*, *block_device_info*, *root_vhd_path*, *eph_vhd_path*, *vm_gen*)

destroy(*instance*, *network_info=None*, *block_device_info=None*, *destroy_disks=True*)

get_console_output(*instance*)

get_image_vm_generation(*root_vhd_path*, *image_meta*)

get_info(*instance*)
Get information about the VM.

list_instance_uuids()

list_instances()

log_vm_serial_output(*instance_name*, *instance_uuid*)

pause(*instance*)
Pause VM instance.

power_off(*instance*, *timeout=0*, *retry_interval=0*)
Power off the specified instance.

power_on(*instance*, *block_device_info=None*)
Power on the specified instance.

reboot(*instance*, *network_info*, *reboot_type*)
Reboot the specified instance.

restart_vm_log_writers()

resume(*instance*)
Resume the suspended VM instance.

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)
Resume guest state when a host is booted.

spawn(**args*, ***kwds*)
Create a new VM and start it.

suspend(*instance*)
Suspend the specified instance.

unpause(*instance*)
Unpause paused VM instance.

check_admin_permissions(*function*)

3.7.666 The nova.virt.hyperv.vmutils Module

Utility class for VM related operations on Hyper-V.

exception HyperVAuthorizationException(*message=None*)
Bases: nova.virt.hyperv.vmutils.HyperVException

exception HyperVException(*message=None*)
Bases: nova.exception.NovaException

exception `UnsupportedConfigDriveFormatException`(*message=None*)
Bases: `nova.virt.hyperv.vmutils.HyperVException`

exception `VHDResizeException`(*message=None*)
Bases: `nova.virt.hyperv.vmutils.HyperVException`

class `VMUtils`(*host='.'*)
Bases: `object`

`attach_drive`(*vm_name*, *path*, *ctrller_path*, *drive_addr*, *drive_type='VHD'*)
Create a drive and attach it to the vm.

`attach_ide_drive`(*vm_name*, *path*, *ctrller_addr*, *drive_addr*, *drive_type='VHD'*)

`attach_scsi_drive`(*vm_name*, *path*, *drive_type='VHD'*)

`attach_volume_to_controller`(*vm_name*, *controller_path*, *address*, *mounted_disk_path*)
Attach a volume to a controller.

`check_admin_permissions`()

`check_ret_val`(*ret_val, job_path, success_values=[0]*)

`create_nic`(*vm_name*, *nic_name*, *mac_address*)
Create a (synthetic) nic and attach it to the vm.

`create_scsi_controller`(*vm_name*)
Create an iscsi controller ready to mount volumes.

`create_vm`(*vm_name*, *memory_mb*, *vcpus_num*, *limit_cpu_features*, *dynamic_memory_ratio*, *vm_gen*, *instance_path*, *notes=None*)
Creates a VM.

`destroy_vm`(*vm_name*)

`detach_vm_disk`(*vm_name*, *disk_path*, *is_physical=True*)

`enable_vm_metrics_collection`(*vm_name*)

`get_active_instances`()
Return the names of all the active instances known to Hyper-V.

`get_attached_disks`(*scsi_controller_path*)

`get_controller_volume_paths`(*controller_path*)

`get_free_controller_slot`(*scsi_controller_path*)

`get_mounted_disk_by_drive_number`(*device_number*)

`get_vm_id`(*vm_name*)

`get_vm_ide_controller`(*vm_name*, *ctrller_addr*)

`get_vm_scsi_controller`(*vm_name*)

`get_vm_serial_port_connection`(*vm_name*, *update_connection=None*)

`get_vm_storage_paths`(*vm_name*)

`get_vm_summary_info`(*vm_name*)

`list_instance_notes`()

`list_instances`()
Return the names of all the instances known to Hyper-V.

`remove_vm_snapshot`(*snapshot_path*)

set_disk_host_resource(*vm_name*, *controller_path*, *address*, *mounted_disk_path*)

set_nic_connection(*vm_name*, *nic_name*, *vswitch_conn_data*)

set_vm_state(*vm_name*, *req_state*)
Set the desired state of the VM.

soft_shutdown_vm(*vm_name*)

take_vm_snapshot(*vm_name*)

update_vm(*vm_name*, *memory_mb*, *vcpus_num*, *limit_cpu_features*, *dynamic_memory_ratio*)

vm_exists(*vm_name*)

3.7.667 The nova.virt.hyperv.vmutilsv2 Module

Utility class for VM related operations. Based on the "root/virtualization/v2" namespace available starting with Hyper-V Server / Windows Server 2012.

class VMUtilsV2(*host='.'*)
Bases: nova.virt.hyperv.vmutils.VMUtils

attach_drive(*vm_name*, *path*, *ctrller_path*, *drive_addr*, *drive_type='VHD'*)
Create a drive and attach it to the vm.

attach_volume_to_controller(*vm_name*, *controller_path*, *address*, *mounted_disk_path*)
Attach a volume to a controller.

create_scsi_controller(*vm_name*)
Create an iscsi controller ready to mount volumes.

destroy_vm(*vm_name*)

enable_vm_metrics_collection(*vm_name*)

get_vm_dvd_disk_paths(*vm_name*)

get_vm_state(*vm_name*)

list_instance_notes()

list_instances()
Return the names of all the instances known to Hyper-V.

remove_vm_snapshot(*snapshot_path*)

set_nic_connection(*vm_name*, *nic_name*, *vswitch_conn_data*)

take_vm_snapshot(*vm_name*)

3.7.668 The nova.virt.hyperv.volumeops Module

Management class for Storage-related functions (attach, detach, etc).

class ISCSIVolumeDriver
Bases: object

attach_volume(*connection_info*, *instance_name*, *ebs_root=False*)
Attach a volume to the SCSI controller or to the IDE controller if ebs_root is True

detach_volume(*connection_info*, *instance_name*)
Detach a volume to the SCSI controller.

disconnect_volumes(*block_device_mapping*)

fix_instance_volume_disk_path(*instance_name*, *connection_info*, *disk_address*)

get_target_from_disk_path(*physical_drive_path*)

get_target_lun_count(*target_iqn*)

initialize_volume_connection(*connection_info*)

login_storage_target(*connection_info*)

logout_storage_target(*target_iqn*, *disconnected_luns_count=1*)

class SMBFSVolumeDriver

Bases: object

attach_volume(*connection_info*, *instance_name*, *ebs_root=False*)

detach_volume(*connection_info*, *instance_name*)

disconnect_volumes(*block_device_mapping*)

ensure_share_mounted(*connection_info*)

fix_instance_volume_disk_path(*instance_name*, *connection_info*, *disk_address*)

initialize_volume_connection(*connection_info*)

class VolumeOps

Bases: object

Management class for Volume-related tasks

attach_volume(*connection_info*, *instance_name*, *ebs_root=False*)

attach_volumes(*block_device_info*, *instance_name*, *ebs_root*)

detach_volume(*connection_info*, *instance_name*)

disconnect_volumes(*block_device_info*)

ebs_root_in_block_devices(*block_device_info*)

fix_instance_volume_disk_paths(*instance_name*, *block_device_info*)

get_volume_connector(*instance*)

initialize_volumes_connection(*block_device_info*)

3.7.669 The nova.virt.hyperv.volumeutils Module

Helper methods for operations related to the management of volumes, and storage repositories

Official Microsoft iSCSI Initiator and iSCSI command line interface documentation can be retrieved at: http://www.microsoft.com/en-us/download/details.aspx?id=34750

class VolumeUtils

Bases: nova.virt.hyperv.basevolumeutils.BaseVolumeUtils

execute(**args*, ***kwargs*)

execute_log_out(*session_id*)

Executes log out of the session described by its session ID.

login_storage_target(*target_lun*, *target_iqn*, *target_portal*, *auth_username=None*, *auth_password=None*)
Ensure that the target is logged in.

logout_storage_target(*target_iqn*)
Logs out storage target through its session id.

3.7.670 The nova.virt.hyperv.volumeutilsv2 Module

Helper methods for operations related to the management of volumes and storage repositories on Windows Server 2012 and above

class VolumeUtilsV2(*host='.'*)
Bases: nova.virt.hyperv.basevolumeutils.BaseVolumeUtils

execute_log_out(*session_id*)

login_storage_target(*target_lun*, *target_iqn*, *target_portal*, *auth_username=None*, *auth_password=None*)
Ensure that the target is logged in.

logout_storage_target(*target_iqn*)
Logs out storage target through its session id.

3.7.671 The nova.virt.image.model Module

class Image(*format*)
Bases: object

Base class for all image types.

All image types have a format, though for many of them only a subset of formats will commonly be used. For example, block devices are almost always going to be FORMAT_RAW. Though it is in fact possible from a technical POV to store a qcow2 data inside a block device, Nova does not (at this time) make use of such possibilities.

class LocalBlockImage(*path*)
Bases: nova.virt.image.model.LocalImage

Class for images that are block devices on the local host

class LocalFileImage(*path*, *format*)
Bases: nova.virt.image.model.LocalImage

Class for images that are files on a locally accessible filesystem

class LocalImage(*path*, *format*)
Bases: nova.virt.image.model.Image

Class for images that are paths within the local filesystem

class RBDImage(*name*, *pool*, *user*, *password*, *servers*)
Bases: nova.virt.image.model.Image

Class for images that are volumes on a remote RBD server

3.7.672 The nova.virt.imagecache Module

class ImageCacheManager
Bases: object

Base class for the image cache manager.

This class will provide a generic interface to the image cache manager.

update (*context*, *all_instances*)
The cache manager.

This will invoke the cache manager. This will update the cache according to the defined cache management scheme. The information populated in the cached stats will be used for the cache management.

3.7.673 The nova.virt.images Module

Handling of VM disk images.

convert_image (*source*, *dest*, *out_format*, *run_as_root=False*)
Convert image to other format.

fetch (*context*, *image_href*, *path*, *_user_id*, *_project_id*, *max_size=0*)

fetch_to_raw (*context*, *image_href*, *path*, *user_id*, *project_id*, *max_size=0*)

get_info (*context*, *image_href*)

qemu_img_info (*path*)
Return an object containing the parsed output from qemu-img info.

3.7.674 The nova.virt.ironic.client_wrapper Module

class IronicClientWrapper
Bases: object

Ironic client wrapper class that encapsulates retry logic.

call (*method*, **args*, ***kwargs*)
Call an Ironic client method and retry on errors.

Parameters

- **method** – Name of the client method to call as a string.
- **args** – Client method arguments.
- **kwargs** – Client method keyword arguments.

Raises NovaException if all retries failed.

3.7.675 The nova.virt.ironic.driver Module

A driver wrapping the Ironic API, such that Nova may provision bare metal resources.

class IronicDriver (*virtapi*, *read_only=False*)
Bases: nova.virt.driver.ComputeDriver

Hypervisor driver for Ironic - bare metal provisioning.

capabilities = {'supports_recreate': False, 'has_imagecache': False, 'supports_migrate_to_same_host': False}

deallocate_networks_on_reschedule(*instance*)
Does the driver want networks deallocated on reschedule?

Parameters **instance** – the instance object.

Returns Boolean value. If True deallocate networks on reschedule.

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)
Destroy the specified instance, if it can be found.

Parameters

- **context** – The security context.
- **instance** – The instance object.
- **network_info** – Instance network information.
- **block_device_info** – Instance block device information. Ignored by this driver.
- **destroy_disks** – Indicates if disks should be destroyed. Ignored by this driver.
- **migrate_data** – implementation specific params. Ignored by this driver.

ensure_filtering_rules_for_instance(*instance*, *network_info*)
Set up filtering rules.

Parameters

- **instance** – The instance object.
- **network_info** – Instance network information.

get_available_nodes(*refresh=False*)
Returns the UUIDs of all nodes in the Ironic inventory.

Parameters **refresh** – Boolean value; If True run update first. Ignored by this driver.

Returns a list of UUIDs

get_available_resource(*nodename*)
Retrieve resource information.

This method is called when nova-compute launches, and as part of a periodic task that records the results in the DB.

Parameters **nodename** – the UUID of the node.

Returns a dictionary describing resources.

get_info(*instance*)
Get the current state and resource usage for this instance.

If the instance is not found this method returns (a dictionary with) NOSTATE and all resources == 0.

Parameters **instance** – the instance object.

Returns a InstanceInfo object

init_host(*host*)
Initialize anything that is necessary for the driver to function.

Parameters **host** – the hostname of the compute host.

instance_exists(*instance*)
Checks the existence of an instance.

Checks the existence of an instance. This is an override of the base method for efficiency.

Parameters **instance** – The instance object.

Returns True if the instance exists. False if not.

list_instance_uuids()
Return the UUIDs of all the instances provisioned.

Returns a list of instance UUIDs.

list_instances()
Return the names of all the instances provisioned.

Returns a list of instance names.

macs_for_instance(*instance*)
List the MAC addresses of an instance.

List of MAC addresses for the node which this instance is associated with.

Parameters **instance** – the instance object.

Returns None, or a set of MAC ids (e.g. set(['12:34:56:78:90:ab'])). None means 'no constraints', a set means 'these and only these MAC addresses'.

node_is_available(*nodename*)
Confirms a Nova hypervisor node exists in the Ironic inventory.

Parameters **nodename** – The UUID of the node.

Returns True if the node exists, False if not.

plug_vifs(*instance*, *network_info*)
Plug VIFs into networks.

Parameters

- **instance** – The instance object.
- **network_info** – Instance network information.

power_off(*instance*, *timeout=0*, *retry_interval=0*)
Power off the specified instance.

NOTE: Ironic does not support soft-off, so this method ignores timeout and retry_interval parameters.

NOTE: Unlike the libvirt driver, this method does not delete and recreate the instance; it preserves local state.

Parameters

- **instance** – The instance object.
- **timeout** – time to wait for node to shutdown. Ignored by this driver.
- **retry_interval** – How often to signal node while waiting for it to shutdown. Ignored by this driver.

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)
Power on the specified instance.

NOTE: Unlike the libvirt driver, this method does not delete and recreate the instance; it preserves local state.

Parameters

- **context** – The security context.
- **instance** – The instance object.
- **network_info** – Instance network information. Ignored by this driver.
- **block_device_info** – Instance block device information. Ignored by this driver.

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)

Reboot the specified instance.

NOTE: Ironic does not support soft-off, so this method always performs a hard-reboot.

NOTE: Unlike the libvirt driver, this method does not delete and recreate the instance; it preserves local state.

Parameters

- **context** – The security context.
- **instance** – The instance object.
- **network_info** – Instance network information. Ignored by this driver.
- **reboot_type** – Either a HARD or SOFT reboot. Ignored by this driver.
- **block_device_info** – Info pertaining to attached volumes. Ignored by this driver.
- **bad_volumes_callback** – Function to handle any bad volumes encountered. Ignored by this driver.

rebuild(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *bdms*, *detach_block_devices*, *attach_block_devices*, *network_info=None*, *recreate=False*, *block_device_info=None*, *preserve_ephemeral=False*)

Rebuild/redeploy an instance.

This version of rebuild() allows for supporting the option to preserve the ephemeral partition. We cannot call spawn() from here because it will attempt to set the instance_uuid value again, which is not allowed by the Ironic API. It also requires the instance to not have an 'active' provision state, but we cannot safely change that. Given that, we implement only the portions of spawn() we need within rebuild().

Parameters

- **context** – The security context.
- **instance** – The instance object.
- **image_meta** – Image object returned by nova.image.glance that defines the image from which to boot this instance. Ignored by this driver.
- **injected_files** – User files to inject into instance. Ignored by this driver.
- **admin_password** – Administrator password to set in instance. Ignored by this driver.
- **bdms** – block-device-mappings to use for rebuild. Ignored by this driver.
- **detach_block_devices** – function to detach block devices. See nova.compute.manager.ComputeManager:_rebuild_default_impl for usage. Ignored by this driver.
- **attach_block_devices** – function to attach block devices. See nova.compute.manager.ComputeManager:_rebuild_default_impl for usage. Ignored by this driver.

- **network_info** – Instance network information. Ignored by this driver.
- **recreate** – Boolean value; if True the instance is recreated on a new hypervisor - all the cleanup of old state is skipped. Ignored by this driver.
- **block_device_info** – Instance block device information. Ignored by this driver.
- **preserve_ephemeral** – Boolean value; if True the ephemeral must be preserved on rebuild.

refresh_instance_security_rules(*instance*)
Refresh security group rules from data store.

Gets called when an instance gets added to or removed from the security group the instance is a member of or if the group gains or loses a rule.

Parameters **instance** – The instance object.

refresh_provider_fw_rules()
Triggers a firewall update based on database changes.

refresh_security_group_members(*security_group_id*)
Refresh security group members from data store.

Invoked when instances are added/removed to a security group.

Parameters **security_group_id** – The security group id.

refresh_security_group_rules(*security_group_id*)
Refresh security group rules from data store.

Invoked when security group rules are updated.

Parameters **security_group_id** – The security group id.

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)
Deploy an instance.

Parameters

- **context** – The security context.
- **instance** – The instance object.
- **image_meta** – Image dict returned by nova.image.glance that defines the image from which to boot this instance.
- **injected_files** – User files to inject into instance.
- **admin_password** – Administrator password to set in instance.
- **network_info** – Instance network information.
- **block_device_info** – Instance block device information. Ignored by this driver.

unfilter_instance(*instance*, *network_info*)
Stop filtering instance.

Parameters

- **instance** – The instance object.
- **network_info** – Instance network information.

unplug_vifs(*instance*, *network_info*)
Unplug VIFs from networks.

Parameters

- **instance** – The instance object.
- **network_info** – Instance network information.

map_power_state(*state*)

3.7.676 The nova.virt.ironic.ironic_states Module

Mapping of bare metal node states.

Setting the node *power_state* is handled by the conductor's power synchronization thread. Based on the power state retrieved from the driver for the node, the state is set to POWER_ON or POWER_OFF, accordingly. Should this fail, the *power_state* value is left unchanged, and the node is placed into maintenance mode.

The *power_state* can also be set manually via the API. A failure to change the state leaves the current state unchanged. The node is NOT placed into maintenance mode in this case.

ACTIVE = 'active'
Node is successfully deployed and associated with an instance.

AVAILABLE = 'available'
Node is available for use and scheduling.

This state is replacing the NOSTATE state used prior to Kilo.

CLEANFAIL = 'clean failed'
Node failed cleaning. This requires operator intervention to resolve.

CLEANING = 'cleaning'
Node is being automatically cleaned to prepare it for provisioning.

DELETED = 'deleted'
Node tear down was successful.

In Juno, target_provision_state was set to this value during node tear down. In Kilo, this will be a transitory value of provision_state, and never represented in target_provision_state.

DELETING = 'deleting'
Node is actively being torn down.

DEPLOYDONE = 'deploy complete'
Node was successfully deployed.

This is mainly a target provision state used during deployment. A successfully deployed node should go to ACTIVE status.

DEPLOYFAIL = 'deploy failed'
Node deployment failed.

DEPLOYING = 'deploying'
Node is ready to receive a deploy request, or is currently being deployed.

A node will have its *provision_state* set to DEPLOYING briefly before it receives its initial deploy request. It will also move to this state from DEPLOYWAIT after the callback is triggered and deployment is continued (disk partitioning and image copying).

DEPLOYWAIT = 'wait call-back'
Node is waiting to be deployed.

This will be the node *provision_state* while the node is waiting for the driver to finish deployment.

ERROR = 'error'
An error occurred during node processing.

The *last_error* attribute of the node details should contain an error message.

INSPECTFAIL = 'inspect failed'
Node inspection failed.

INSPECTING = 'inspecting'
Node is under inspection. This is the provision state used when inspection is started. A successfully inspected node shall transition to MANAGEABLE status.

MANAGEABLE = 'manageable'
Node is in a manageable state. This state indicates that Ironic has verified, at least once, that it had sufficient information to manage the hardware. While in this state, the node is not available for provisioning (it must be in the AVAILABLE state for that).

NOSTATE = None
No state information.

This state is used with power_state to represent a lack of knowledge of power state, and in target_*_state fields when there is no target.

Prior to the Kilo release, Ironic set node.provision_state to NOSTATE when the node was available for provisioning. During Kilo cycle, this was changed to the AVAILABLE state.

POWER_OFF = 'power off'
Node is powered off.

POWER_ON = 'power on'
Node is powered on.

REBOOT = 'rebooting'
Node is rebooting.

REBUILD = 'rebuild'
Node is to be rebuilt. This is not used as a state, but rather as a "verb" when changing the node's provision_state via the REST API.

3.7.677 The nova.virt.ironic.patcher Module

Helper classes for Ironic HTTP PATCH creation.

class GenericDriverFields(*node*)
Bases: object

get_cleanup_patch(*instance*, *network_info*, *flavor*)
Build a patch to clean up the fields.

Parameters

- **instance** – the instance object.
- **network_info** – the instance network information.
- **flavor** – the flavor object.

Returns a json-patch with the fields that needs to be updated.

get_deploy_patch(*instance*, *image_meta*, *flavor*, *preserve_ephemeral=None*)
Build a patch to add the required fields to deploy a node.

Parameters

- **instance** – the instance object.
- **image_meta** – the metadata associated with the instance image.
- **flavor** – the flavor object.
- **preserve_ephemeral** – preserve_ephemeral status (bool) to be specified during rebuild.

Returns a json-patch with the fields that needs to be updated.

create(*node*)

Create an instance of the appropriate DriverFields class.

Parameters **node** – a node object returned from ironicclient

Returns A GenericDriverFields instance.

3.7.678 The nova.virt.libvirt.blockinfo Module

Handling of block device information and mapping.

This module contains helper methods for interpreting the block device information and determining the suitable mapping to guest devices and libvirt XML.

Throughout these methods there are a number of standard variables / types used

- 'mapping': a dict contains the storage device mapping.

  For the default disk types it will contain the following keys & values:

  'disk' -> disk_info 'disk.rescue' -> disk_info 'disk.local' -> disk_info 'disk.swap' -> disk_info 'disk.config' -> disk_info

  If any of the default disks are overridden by the block device info mappings, the hash value will be None

  For any ephemeral device there will also be a dict entry

  'disk.eph$NUM' -> disk_info

  For any volume device there will also be a dict entry:

  $path -> disk_info

  Finally a special key will refer to the root device:

  'root' -> disk_info

- 'disk_info': a tuple specifying disk configuration

  It contains the following 3 fields

  (disk bus, disk dev, device type)

  and possibly these optional fields: ('format',)

- 'disk_bus': the guest bus type ('ide', 'virtio', 'scsi', etc)
- 'disk_dev': the device name 'vda', 'hdc', 'sdf', 'xvde' etc
- 'device_type': type of device eg 'disk', 'cdrom', 'floppy'
- 'format': Which format to apply to the device if applicable
- 'boot_index': Number designating the boot order of the device

default_device_names(*virt_type*, *context*, *instance*, *block_device_info*, *image_meta*)

find_disk_dev_for_disk_bus(*mapping*, *bus*, *last_device=False*)
Identify a free disk dev name for a bus.

Determines the possible disk dev names for the bus, and then checks them in order until it identifies one that is not yet used in the disk mapping. If 'last_device' is set, it will only consider the last available disk dev name.

Returns the chosen disk_dev name, or raises an exception if none is available.

get_boot_order(*disk_info*)

get_config_drive_type(*os_type=None*)
Determine the type of config drive.

Config drive will be: 'cdrom' in case of config_drive is set to iso9660; 'disk' in case of config_drive is set to vfat; 'fs' in case of os_type is EXE and virt_type is parallels; Autodetected from (cdrom, disk, fs) in case of config_drive is None; Otherwise, an exception of unknown format will be thrown.

Returns a string indicating the config drive type.

get_dev_count_for_disk_bus(*disk_bus*)
Determine the number disks supported.

Determine how many disks can be supported in a single VM for a particular disk bus.

Returns the number of disks supported.

get_dev_prefix_for_disk_bus(*disk_bus*)
Determine the dev prefix for a disk bus.

Determine the dev prefix to be combined with a disk number to fix a disk_dev. eg 'hd' for 'ide' bus can be used to form a disk dev 'hda'

Returns the dev prefix or raises an exception if the disk bus is unknown.

get_device_name(*bdm*)
Get the device name if present regardless of the bdm format.

get_disk_bus_for_device_type(*virt_type*, *image_meta*, *device_type='disk'*, *instance=None*)
Determine the best disk bus to use for a device type.

Considering the currently configured virtualization type, return the optimal disk_bus to use for a given device type. For example, for a disk on KVM it will return 'virtio', while for a CDROM it will return 'ide' on x86_64 and 'scsi' on ppc64.

Returns the disk_bus, or returns None if the device type is not supported for this virtualization

get_disk_bus_for_disk_dev(*virt_type*, *disk_dev*)
Determine the disk bus for a disk device.

Given a disk device like 'hda', 'sdf', 'xvdb', etc guess what the most appropriate disk bus is for the currently configured virtualization technology

Returns the disk bus, or raises an Exception if the disk device prefix is unknown.

get_disk_info(*virt_type*, *instance*, *image_meta*, *block_device_info=None*, *rescue=False*)
Determine guest disk mapping info.

This is a wrapper around get_disk_mapping, which also returns the chosen disk_bus and cdrom_bus. The returned data is in a dict

- disk_bus: the bus for harddisks
- cdrom_bus: the bus for CDROMs
- mapping: the disk mapping

Returns the disk mapping disk.

get_disk_mapping(*virt_type*, *instance*, *disk_bus*, *cdrom_bus*, *image_meta*, *block_device_info=None*, *rescue=False*)
Determine how to map default disks to the virtual machine.

This is about figuring out whether the default 'disk', 'disk.local', 'disk.swap' and 'disk.config' images have been overridden by the block device mapping.

Returns the guest disk mapping for the devices.

get_eph_disk(*index*)

get_info_from_bdm(*virt_type*, *image_meta*, *bdm*, *mapping=None*, *disk_bus=None*, *dev_type=None*, *allowed_types=None*, *assigned_devices=None*)

get_next_disk_info(*mapping*, *disk_bus*, *device_type='disk'*, *last_device=False*, *boot_index=None*)
Determine the disk info for the next device on disk_bus.

Considering the disks already listed in the disk mapping, determine the next available disk dev that can be assigned for the disk bus.

Returns the disk_info for the next available disk.

get_root_info(*virt_type*, *image_meta*, *root_bdm*, *disk_bus*, *cdrom_bus*, *root_device_name=None*)

has_default_ephemeral(*instance*, *disk_bus*, *block_device_info*, *mapping*)

has_disk_dev(*mapping*, *disk_dev*)
Determine if a disk device name has already been used.

Looks at all the keys in mapping to see if any corresponding disk_info tuple has a device name matching disk_dev

Returns True if the disk_dev is in use.

is_disk_bus_valid_for_virt(*virt_type*, *disk_bus*)

update_bdm(*bdm*, *info*)

3.7.679 The nova.virt.libvirt.compat Module

get_domain_info(*libvirt*, *host*, *virt_dom*)
Method virDomain.info (libvirt version < 1.2.11) is affected by a race condition. See bug #1372670 for more details. This method detects it to perform a retry.

3.7.680 The nova.virt.libvirt.config Module

Configuration for libvirt objects.

Classes to represent the configuration of various libvirt objects and support conversion to/from XML. These classes are solely concerned by providing direct Object <-> XML document conversions. No policy or operational decisions should be made by code in these classes. Such policy belongs in the 'designer.py' module which provides simplified helpers for populating up config object instances.

class LibvirtConfigCPU(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

add_feature(*feat*)

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCPUFeature**(*name=None*, ***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCaps**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCapsGuest**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCapsHost**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCapsNUMACPU**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCapsNUMACell**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCapsNUMAPages**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigCapsNUMATopology**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuest**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

add_device(*dev*)

format_dom()

parse_dom(*xmldoc*)

set_clock(*clk*)

class **LibvirtConfigGuestCPU**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigCPU

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestCPUFeature**(*name=None*, ***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigCPUFeature

format_dom()

class **LibvirtConfigGuestCPUNUMA**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestCPUNUMACell**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestCPUTune**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestCPUTuneEmulatorPin**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestCPUTuneVCPUPin**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestChannel**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestCharBase

format_dom()

class **LibvirtConfigGuestChar**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestCharBase

format_dom()

class **LibvirtConfigGuestCharBase**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class **LibvirtConfigGuestClock**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

add_timer(*tm*)

format_dom()

class **LibvirtConfigGuestConsole**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestChar

class **LibvirtConfigGuestController**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class **LibvirtConfigGuestDevice**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

class **LibvirtConfigGuestDisk**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestDiskBackingStore**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

parse_dom(*xmldoc*)

class **LibvirtConfigGuestFeature**(*name*, ***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

class **LibvirtConfigGuestFeatureACPI**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestFeature

class **LibvirtConfigGuestFeatureAPIC**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestFeature

class **LibvirtConfigGuestFeatureHyperV**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestFeature

MIN_SPINLOCK_RETRIES = 4095

format_dom()

class **LibvirtConfigGuestFeaturePAE**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestFeature

class **LibvirtConfigGuestFilesys**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class **LibvirtConfigGuestGIDMap**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestIDMap

class **LibvirtConfigGuestGraphics**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class **LibvirtConfigGuestHostdev**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestHostdevPCI**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestHostdev

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestIDMap**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

parse_dom(*xmldoc*)

class **LibvirtConfigGuestInput**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class **LibvirtConfigGuestInterface**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

add_filter_param(*key*, *value*)

add_vport_param(*key*, *value*)

format_dom()

class **LibvirtConfigGuestMemoryBacking**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestMemoryBackingPage**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestMemoryTune**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestMetaNovaFlavor**
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestMetaNovaInstance**
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestMetaNovaOwner**
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestNUMATune**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestNUMATuneMemNode**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class **LibvirtConfigGuestNUMATuneMemory**(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class LibvirtConfigGuestRng(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class LibvirtConfigGuestSMBIOS(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class LibvirtConfigGuestSerial(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestChar

class LibvirtConfigGuestSnapshot(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

add_disk(*disk*)

format_dom()

class LibvirtConfigGuestSnapshotDisk(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

Disk class for handling disk information in snapshots.

Similar to LibvirtConfigGuestDisk, but used to represent disk entities in <domainsnapshot> structures rather than real devices. These typically have fewer members, and different expectations for which fields are required.

format_dom()

parse_dom(*xmldoc*)

class LibvirtConfigGuestSysinfo(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class LibvirtConfigGuestTimer(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class LibvirtConfigGuestUIDMap(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestIDMap

class LibvirtConfigGuestVideo(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class LibvirtConfigGuestWatchdog(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class LibvirtConfigMemoryBalloon(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigGuestDevice

format_dom()

class LibvirtConfigNodeDevice(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

Libvirt Node Devices parser.

parse_dom(*xmldoc*)

class LibvirtConfigNodeDevicePciCap(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

Libvirt Node Devices pci capability parser.

parse_dom(*xmldoc*)

class LibvirtConfigNodeDevicePciSubFunctionCap(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

parse_dom(*xmldoc*)

class LibvirtConfigObject(***kwargs*)
Bases: object

format_dom()

parse_dom(*xmldoc*)

parse_str(*xmlstr*)

to_xml(*pretty_print=True*)

class LibvirtConfigSeclabel(***kwargs*)
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

class LibvirtConfigSecret
Bases: nova.virt.libvirt.config.LibvirtConfigObject

format_dom()

get_yes_no_str(*value*)

3.7.681 The nova.virt.libvirt.designer Module

Policy based configuration of libvirt objects

This module provides helper APIs for populating the config.py classes based on common operational needs / policies

set_vif_bandwidth_config(*conf*, *inst_type*)
Config vif inbound/outbound bandwidth limit. parameters are set in instance_type_extra_specs table, key is in the format quota:vif_inbound_average.

set_vif_guest_frontend_config(*conf*, *mac*, *model*, *driver*)
Populate a LibvirtConfigGuestInterface instance with guest frontend details.

set_vif_host_backend_802qbg_config(*conf*, *devname*, *managerid*, *typeid*, *typeidversion*, *instanceid*, *tapname=None*)
Populate a LibvirtConfigGuestInterface instance with host backend details for an 802.1qbg device.

set_vif_host_backend_802qbh_config(*conf*, *net_type*, *devname*, *profileid*, *tapname=None*)
Populate a LibvirtConfigGuestInterface instance with host backend details for an 802.1qbh device.

set_vif_host_backend_bridge_config(*conf*, *brname*, *tapname=None*)
Populate a LibvirtConfigGuestInterface instance with host backend details for a software bridge.

set_vif_host_backend_direct_config(*conf*, *devname*)
Populate a LibvirtConfigGuestInterface instance with direct Interface.

set_vif_host_backend_ethernet_config(*conf*, *tapname*)
Populate a LibvirtConfigGuestInterface instance with host backend details for an externally configured host device.

NB use of this configuration is discouraged by libvirt project and will mark domains as 'tainted'.

set_vif_host_backend_hw_veb(*conf*, *net_type*, *devname*, *vlan*, *tapname=None*)
Populate a LibvirtConfigGuestInterface instance with host backend details for an device that supports hardware virtual ethernet bridge.

set_vif_host_backend_ovs_config(*conf*, *brname*, *interfaceid*, *tapname=None*)
Populate a LibvirtConfigGuestInterface instance with host backend details for an OpenVSwitch bridge.

set_vif_host_backend_vhostuser_config(*conf*, *mode*, *path*)
Populate a LibvirtConfigGuestInterface instance with host backend details for vhostuser socket.

3.7.682 The nova.virt.libvirt.dmcrypt Module

create_volume(*target*, *device*, *cipher*, *key_size*, *key*)
Sets up a dmcrypt mapping

Parameters

- **target** – device mapper logical device name
- **device** – underlying block device
- **cipher** – encryption cipher string digestible by cryptsetup
- **key_size** – encryption key size
- **key** – encryption key as an array of unsigned bytes

delete_volume(*target*)
Deletes a dmcrypt mapping

Parameters **target** – name of the mapped logical device

is_encrypted(*path*)
Returns true if the path corresponds to an encrypted disk.

list_volumes()
Function enumerates encrypted volumes.

volume_name(*base*)
Returns the suffixed dmcrypt volume name.

This is to avoid collisions with similarly named device mapper names for LVM volumes

3.7.683 The nova.virt.libvirt.driver Module

A connection to a hypervisor through libvirt.

Supports KVM, LXC, QEMU, UML, XEN and Parallels.

class GuestNumaConfig
Bases: `tuple`

GuestNumaConfig(cpuset, cputune, numaconfig, numatune)

cpuset
Alias for field number 0

cputune
Alias for field number 1

numaconfig
Alias for field number 2

numatune
Alias for field number 3

class LibvirtDriver(*virtapi*, *read_only=False*)
Bases: nova.virt.driver.ComputeDriver

attach_interface(*instance*, *image_meta*, *vif*)

attach_volume(*context*, *connection_info*, *instance*, *mountpoint*, *disk_bus=None*, *device_type=None*, *encryption=None*)

block_stats(*instance*, *disk_id*)
Note that this function takes an instance name.

capabilities = {'supports_recreate': True, 'has_imagecache': True, 'supports_migrate_to_same_host': False}

check_can_live_migrate_destination(*context*, *instance*, *src_compute_info*, *dst_compute_info*, *block_migration=False*, *disk_over_commit=False*)
Check if it is possible to execute live migration.

This runs checks on the destination host, and then calls back to the source host to check the results.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance
- **block_migration** – if true, prepare for block migration
- **disk_over_commit** – if true, allow disk over commit

Returns a dict containing: :filename: name of the tmpfile under CONF.instances_path :block_migration: whether this is block migration :disk_over_commit: disk-over-commit factor on dest host :disk_available_mb: available disk space on dest host

check_can_live_migrate_destination_cleanup(*context*, *dest_check_data*)
Do required cleanup on dest host after check_can_live_migrate calls

Parameters **context** – security context

check_can_live_migrate_source(*context*, *instance*, *dest_check_data*, *block_device_info=None*)
Check if it is possible to execute live migration.

This checks if the live migration can succeed, based on the results from check_can_live_migrate_destination.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance
- **dest_check_data** – result of check_can_live_migrate_destination
- **block_device_info** – result of _get_instance_block_device_info

Returns a dict containing migration info

check_instance_shared_storage_cleanup(*context*, *data*)

check_instance_shared_storage_local(*context*, *instance*)
Check if instance files located on shared storage.

This runs check on the destination host, and then calls back to the source host to check the results.

Parameters

- **context** – security context
- **instance** – nova.objects.instance.Instance object

Returns

- **tempfile: A dict containing the tempfile info on the destination** host
- None:
  1. If the instance path is not existing.
  2. If the image backend is shared block storage type.

check_instance_shared_storage_remote(*context*, *data*)

cleanup(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*, *destroy_vifs=True*)

confirm_migration(*migration*, *instance*, *network_info*)
Confirms a resize, destroying the source VM.

default_device_names_for_instance(*instance*, *root_device_name*, **block_device_lists*)

default_root_device_name(*instance*, *image_meta*, *root_bdm*)

delete_instance_files(*instance*)

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)

detach_interface(*instance*, *vif*)

detach_volume(*connection_info*, *instance*, *mountpoint*, *encryption=None*)

disk_cachemode

ensure_filtering_rules_for_instance(*instance*, *network_info*)
Ensure that an instance's filtering rules are enabled.

When migrating an instance, we need the filtering rules to be configured on the destination host before starting the migration.

Also, when restarting the compute service, we need to ensure that filtering rules exist for all running services.

filter_defer_apply_off()

filter_defer_apply_on()

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)

get_all_volume_usage(*context*, *compute_host_bdms*)
Return usage info for volumes attached to vms on a given host.

get_available_nodes(*refresh=False*)

get_available_resource(*nodename*)
Retrieve resource information.

This method is called when nova-compute launches, and as part of a periodic task that records the results in the DB.

Parameters **nodename** – will be put in PCI device

Returns dictionary containing resource info

get_console_output(*context*, *instance*)

get_console_pool_info(*console_type*)

get_diagnostics(*instance*)

get_host_cpu_stats()
Return the current CPU state of the host.

get_host_ip_addr()

get_host_uptime()
Returns the result of calling "uptime".

get_info(*instance*)
Retrieve information from libvirt for a specific instance name.

If a libvirt error is encountered during lookup, we might raise a NotFound exception or Error exception depending on how severe the libvirt error is.

get_instance_diagnostics(*instance*)

get_instance_disk_info(*instance*, *block_device_info=None*)

get_serial_console(*context*, *instance*)

get_spice_console(*context*, *instance*)

get_vnc_console(*context*, *instance*)

get_volume_connector(*instance*)

init_host(*host*)

inject_network_info(*instance*, *nw_info*)

instance_exists(*instance*)
Efficient override of base instance_exists method.

instance_on_disk(*instance*)

is_supported_fs_format(*fs_type*)

list_instance_uuids()

list_instances()

live_migration(*context*, *instance*, *dest*, *post_method*, *recover_method*, *block_migration=False*, *migrate_data=None*)
Spawning live_migration operation for distributing high-load.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance object instance object that is migrated.
- **dest** – destination host

- **post_method** – post operation method. expected nova.compute.manager._post_live_migration.
- **recover_method** – recovery method when any exception occurs. expected nova.compute.manager._rollback_live_migration.
- **block_migration** – if true, do block migration.
- **migrate_data** – implementation specific params

manage_image_cache(*context*, *all_instances*)
Manage the local cache of images.

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)

need_legacy_block_device_info

pause(*instance*)
Pause VM instance.

plug_vifs(*instance*, *network_info*)
Plug VIFs into networks.

poll_rebooting_instances(*timeout*, *instances*)

post_live_migration(*context*, *instance*, *block_device_info*, *migrate_data=None*)

post_live_migration_at_destination(*context*, *instance*, *network_info*, *block_migration=False*, *block_device_info=None*)
Post operation of live migration at destination host.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance object instance object that is migrated.
- **network_info** – instance network information
- **block_migration** – if true, post operation of block_migration.

post_live_migration_at_source(*context*, *instance*, *network_info*)
Unplug VIFs from networks at source.

Parameters

- **context** – security context
- **instance** – instance object reference
- **network_info** – instance network information

power_off(*instance*, *timeout=0*, *retry_interval=0*)
Power off the specified instance.

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)
Power on the specified instance.

pre_live_migration(*context*, *instance*, *block_device_info*, *network_info*, *disk_info*, *migrate_data=None*)
Preparation live migration.

quiesce(*context*, *instance*, *image_meta*)
Freeze the guest filesystems to prepare for snapshot.

The qemu-guest-agent must be setup to execute fsfreeze.

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)
Reboot a virtual machine, given an instance reference.

refresh_instance_security_rules(*instance*)

refresh_provider_fw_rules()

refresh_security_group_members(*security_group_id*)

refresh_security_group_rules(*security_group_id*)

rescue(*context*, *instance*, *network_info*, *image_meta*, *rescue_password*)
Loads a VM using rescue images.

A rescue is normally performed when something goes wrong with the primary images and data needs to be corrected/recovered. Rescuing should not edit or over-ride the original image, only allow for data recovery.

resume(*context*, *instance*, *network_info*, *block_device_info=None*)
resume the specified instance.

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)
resume guest state when a host is booted.

rollback_live_migration_at_destination(*context*, *instance*, *network_info*, *block_device_info*, *destroy_disks=True*, *migrate_data=None*)
Clean up destination node after a failed live migration.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Create snapshot from a running VM instance.

This command only works with qemu 0.14+

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)

suspend(*context*, *instance*)
Suspend the specified instance.

swap_volume(*old_connection_info*, *new_connection_info*, *instance*, *mountpoint*, *resize_to*)

unfilter_instance(*instance*, *network_info*)
See comments of same method in firewall_driver.

unpause(*instance*)
Unpause paused VM instance.

unplug_vifs(*instance*, *network_info*)

unquiesce(*context*, *instance*, *image_meta*)
Thaw the guest filesystems after snapshot.

unrescue(*instance*, *network_info*)
Reboot the VM which is being rescued back into primary images.

volume_snapshot_create(*context*, *instance*, *volume_id*, *create_info*)
Create snapshots of a Cinder volume via libvirt.

Parameters

- **instance** – VM instance object reference
- **volume_id** – id of volume being snapshotted

- **create_info** – dict of information used to create snapshots - snapshot_id : ID of snapshot - type : qcow2 / <other> - new_file : qcow2 file created by Cinder which becomes the VM's active image after the snapshot is complete

volume_snapshot_delete(*context*, *instance*, *volume_id*, *snapshot_id*, *delete_info*)

patch_tpool_proxy()
eventlet.tpool.Proxy doesn't work with old-style class in __str__() or __repr__() calls. See bug #962840 for details. We perform a monkey patch to replace those two instance methods.

3.7.684 The nova.virt.libvirt.firewall Module

class IptablesFirewallDriver(*virtapi*, *execute=None*, ***kwargs*)
Bases: nova.virt.firewall.IptablesFirewallDriver

apply_instance_filter(*instance*, *network_info*)
No-op. Everything is done in prepare_instance_filter.

instance_filter_exists(*instance*, *network_info*)
Check nova-instance-instance-xxx exists.

setup_basic_filtering(*instance*, *network_info*)
Set up provider rules and basic NWFilter.

unfilter_instance(*instance*, *network_info*)

class NWFilterFirewall(*virtapi*, *host*, ***kwargs*)
Bases: nova.virt.firewall.FirewallDriver

This class implements a network filtering mechanism by using libvirt's nwfilter. all instances get a filter ("nova-base") applied. This filter provides some basic security such as protection against MAC spoofing, IP spoofing, and ARP spoofing.

apply_instance_filter(*instance*, *network_info*)
No-op. Everything is done in prepare_instance_filter.

get_base_filter_list(*instance*, *allow_dhcp*)
Obtain a list of base filters to apply to an instance. The return value should be a list of strings, each specifying a filter name. Subclasses can override this function to add additional filters as needed. Additional filters added to the list must also be correctly defined within the subclass.

instance_filter_exists(*instance*, *network_info*)
Check nova-instance-instance-xxx exists.

nova_dhcp_filter()
The standard allow-dhcp-server filter is an <ip> one, so it uses ebtables to allow traffic through. Without a corresponding rule in iptables, it'll get blocked anyway.

nova_no_nd_reflection_filter()
This filter protects false positives on IPv6 Duplicate Address Detection(DAD).

setup_basic_filtering(*instance*, *network_info*)
Set up basic filtering (MAC, IP, and ARP spoofing protection).

unfilter_instance(*instance*, *network_info*)
Clear out the nwfilter rules.

3.7.685 The nova.virt.libvirt.guest Module

Manages information about the guest.

This class encapsulates libvirt domain provides certain higher level APIs around the raw libvirt API. These APIs are then used by all the other libvirt related classes

class BlockDevice(*guest*, *disk*)
Bases: object

Wrapper around block device API

COMMIT_DEFAULT_BANDWIDTH = 0

REBASE_DEFAULT_BANDWIDTH = 0

abort_job(*async=False*, *pivot=False*)
Request to cancel any job currently running on the block.

Parameters

- **async** – Request only, do not wait for completion
- **pivot** – Pivot to new file when ending a copy or active commit job

commit(*base*, *top*, *relative=False*)
Commit on block device

For performance during live snapshot it will reduces the disk chain to a single disk.

Parameters **relative** – Keep backing chain referenced using relative names

get_job_info()
Returns information about job currently running

Returns BlockDeviceJobInfo or None

rebase(*base*, *shallow=False*, *reuse_ext=False*, *copy=False*, *relative=False*)
Rebases block to new base

Parameters

- **shallow** – Limit copy to top of source backing chain
- **reuse_ext** – Reuse existing external file of a copy
- **copy** – Start a copy job
- **relative** – Keep backing chain referenced using relative names

resize(*size_kb*)
Resizes block device to Kib size.

class BlockDeviceJobInfo(*job*, *bandwidth*, *cur*, *end*)
Bases: object

class Guest(*domain*)
Bases: object

attach_device(*conf*, *persistent=False*, *live=False*)
Attaches device to the guest.

Parameters

- **conf** – A LibvirtConfigObject of the device to attach
- **persistent** – A bool to indicate whether the change is persistent or not
- **live** – A bool to indicate whether it affect the guest in running state

classmethod create(*xml*, *host*)
Create a new Guest

Parameters

- **xml** – XML definition of the domain to create
- **host** – host.Host connection to define the guest on

Returns guest.Guest Guest ready to be launched

delete_configuration()
Undefines a domain from hypervisor.

detach_device(*conf*, *persistent=False*, *live=False*)
Detaches device to the guest.

Parameters

- **conf** – A LibvirtConfigObject of the device to detach
- **persistent** – A bool to indicate whether the change is persistent or not
- **live** – A bool to indicate whether it affect the guest in running state

enable_hairpin()
Enables hairpin mode for this guest.

get_block_device(*disk*)
Returns a block device wrapper for disk.

get_disk(*device*)
Returns the disk mounted at device

Returns LivirtConfigGuestDisk mounted at device or None

get_interfaces()
Returns a list of all network interfaces for this domain.

get_vcpus_info()
Returns virtual cpus information of guest.

Returns guest.VCPUInfo

get_xml_desc(*dump_inactive=False*, *dump_sensitive=False*, *dump_migratable=False*)
Returns xml description of guest.

Parameters

- **dump_inactive** – Dump inactive domain information
- **dump_sensitive** – Dump security sensitive information
- **dump_migratable** – Dump XML suitable for migration

Returns string XML description of the guest

has_persistent_configuration()
Whether domain config is persistently stored on the host.

id

launch(*pause=False*)
Starts a created guest.

Parameters **pause** – Indicates whether to start and pause the guest

name

poweroff()
Stops a running guest.

resume()
Resumes a suspended guest.

save_memory_state()
Saves the domain's memory state. Requires running domain.

raises: raises libvirtError on error

uuid

class VCPUInfo(*id*, *cpu*, *state*, *time*)
Bases: object

3.7.686 The nova.virt.libvirt.host Module

Manages information about the host OS and hypervisor.

This class encapsulates a connection to the libvirt daemon and provides certain higher level APIs around the raw libvirt API. These APIs are then used by all the other libvirt related classes

class DomainJobInfo(**kwargs*)
Bases: object

Information about libvirt background jobs

This class encapsulates information about libvirt background jobs. It provides a mapping from either the old virDomainGetJobInfo API which returned a fixed list of fields, or the modern virDomainGetJobStats which returns an extendable dict of fields.

classmethod for_domain(*dom*)
Get job info for the domain

Query the libvirt job info for the domain (ie progress of migration, or snapshot operation)

Returns: a DomainJobInfo instance

class Host(*uri*, *read_only=False*, *conn_event_handler=None*, *lifecycle_event_handler=None*)
Bases: object

compare_cpu(*xmlDesc*, *flags=0*)
Compares the given CPU description with the host CPU.

create_secret(*usage_type*, *usage_id*, *password=None*)
Create a secret.

Parameters

- **usage_type** – one of 'iscsi', 'ceph', 'rbd' or 'volume' 'rbd' will be converted to 'ceph'.
- **usage_id** – name of resource in secret
- **password** – optional secret value to set

delete_secret(*usage_type*, *usage_id*)
Delete a secret.

usage_type: one of 'iscsi', 'ceph', 'rbd' or 'volume' usage_id: name of resource in secret

device_lookup_by_name(*name*)
Lookup a node device by its name.

Returns a virNodeDevice instance

find_secret(*usage_type*, *usage_id*)
Find a secret.

usage_type: one of 'iscsi', 'ceph', 'rbd' or 'volume' usage_id: name of resource in secret

get_capabilities()
Returns the host capabilities information

Returns an instance of config.LibvirtConfigCaps representing the capabilities of the host.

Note: The result is cached in the member attribute _caps.

Returns a config.LibvirtConfigCaps object

get_connection()
Returns a connection to the hypervisor

This method should be used to create and return a well configured connection to the hypervisor.

Returns a libvirt.virConnect object

get_cpu_count()
Returns the total numbers of cpu in the host.

get_cpu_stats()
Returns the current CPU state of the host with frequency.

get_domain(*instance*)
Retrieve libvirt domain object for an instance.

Parameters **instance** – an nova.objects.Instance object

Attempt to lookup the libvirt domain objects corresponding to the Nova instance, based on its name. If not found it will raise an exception.InstanceNotFound exception. On other errors, it will raise a exception.NovaException exception.

Returns a libvirt.Domain object

get_domain_info(*virt_dom*)

get_driver_type()
Get hypervisor type.

Returns hypervisor type (ex. qemu)

get_guest(*instance*)
Retrieve libvirt domain object for an instance.

Parameters **instance** – an nova.objects.Instance object

Returns a nova.virt.libvirt.Guest object

get_hostname()
Returns the hostname of the hypervisor.

get_memory_mb_total()
Get the total memory size(MB) of physical computer.

Returns the total amount of memory(MB).

get_memory_mb_used()
Get the used memory size(MB) of physical computer.

Returns the total usage of memory(MB).

get_online_cpus()
Get the set of CPUs that are online on the host

Method is only used by NUMA code paths which check on libvirt version >= 1.0.4. getCPUMap() was introduced in libvirt 1.0.0.

Returns set of online CPUs, raises libvirtError on error

get_version()
Get hypervisor version.

Returns hypervisor version (ex. 12003)

has_min_version(*lv_ver=None*, *hv_ver=None*, *hv_type=None*)

has_version(*lv_ver=None*, *hv_ver=None*, *hv_type=None*)

initialize()

list_instance_domains(*only_running=True*, *only_guests=True*)
Get a list of libvirt.Domain objects for nova instances

Parameters

- **only_running** – True to only return running instances
- **only_guests** – True to filter out any host domain (eg Dom-0)

Query libvirt to a get a list of all libvirt.Domain objects that correspond to nova instances. If the only_running parameter is true this list will only include active domains, otherwise inactive domains will be included too. If the only_guests parameter is true the list will have any "host" domain (aka Xen Domain-0) filtered out.

Returns list of libvirt.Domain objects

list_pci_devices(*flags=0*)
Lookup pci devices.

Returns a list of virNodeDevice instance

write_instance_config(*xml*)
Defines a domain, but does not start it.

Parameters **xml** – XML domain definition of the guest.

Returns a virDomain instance

3.7.687 The nova.virt.libvirt.imagebackend Module

class Backend(*use_cow*)
Bases: object

backend(*image_type=None*)

image(*instance*, *disk_name*, *image_type=None*)
Constructs image for selected backend

Instance Instance name.

Name Image name.

Image_type Image type. Optional, is CONF.libvirt.images_type by default.

snapshot(*instance*, *disk_path*, *image_type=None*)
Returns snapshot for given image

Path path to image

Image_type type of image

class Image(*source_type*, *driver_format*, *is_block_dev=False*)
Bases: object

SUPPORTS_CLONE = False

cache(*fetch_func*, *filename*, *size=None*, **args*, ***kwargs*)
Creates image from template.

Ensures that template and image not already exists. Ensures that base directory exists. Synchronizes on template fetching.

Fetch_func Function that creates the base image Should accept *target* argument.

Filename Name of the file in the image directory

Size Size of created image in bytes (optional)

check_image_exists()

clone(*context*, *image_id_or_uri*)
Clone an image.

Note that clone operation is backend-dependent. The backend may ask the image API for a list of image "locations" and select one or more of those locations to clone an image from.

Parameters image_id_or_uri – The ID or URI of an image to clone.

Raises exception.ImageUnacceptable if it cannot be cloned

create_image(*prepare_template*, *base*, *size*, **args*, ***kwargs*)
Create image from template.

Contains specific behavior for each image type.

Prepare_template function, that creates template. Should accept *target* argument.

Base Template name

Size Size of created image in bytes

get_disk_size(*name*)

get_model(*connection*)
Get the image information model

Returns an instance of nova.virt.image.model.Image

static is_file_in_instance_path()
True if the backend stores images in files under instance path.

static is_shared_block_storage()
True if the backend puts images on a shared block storage.

libvirt_fs_info(*target*, *driver_type=None*)
Get *LibvirtConfigGuestFilesys* filled for this image.

Target target directory inside a container.

Driver_type filesystem driver type, can be loop nbd or ploop.

libvirt_info(*disk_bus*, *disk_dev*, *device_type*, *cache_mode*, *extra_specs*, *hypervisor_version*)
Get *LibvirtConfigGuestDisk* filled for this image.

Disk_dev Disk bus device name

Disk_bus Disk bus type

Device_type Device type for this image.

Cache_mode Caching mode for this image

Extra_specs Instance type extra specs dict.

Hypervisor_version the hypervisor version

resolve_driver_format()
Return the driver format for self.path.

First checks self.disk_info_path for an entry. If it's not there, calls self._get_driver_format(), and then stores the result in self.disk_info_path

See https://bugs.launchpad.net/nova/+bug/1221190

snapshot_extract(*target*, *out_format*)

verify_base_size(*base*, *size*, *base_size=0*)
Check that the base image is not larger than size. Since images can't be generally shrunk, enforce this constraint taking account of virtual image size.

class Lvm(*instance=None*, *disk_name=None*, *path=None*)
Bases: nova.virt.libvirt.imagebackend.Image

create_image(*prepare_template*, *base*, *size*, **args*, ***kwargs*)

static escape(*filename*)

get_model(*connection*)

remove_volume_on_error(**args*, ***kwds*)

snapshot_extract(*target*, *out_format*)

class Ploop(*instance=None*, *disk_name=None*, *path=None*)
Bases: nova.virt.libvirt.imagebackend.Image

create_image(*prepare_template*, *base*, *size*, **args*, ***kwargs*)

class Qcow2(*instance=None*, *disk_name=None*, *path=None*)
Bases: nova.virt.libvirt.imagebackend.Image

create_image(*prepare_template*, *base*, *size*, **args*, ***kwargs*)

get_model(*connection*)

static is_file_in_instance_path()

snapshot_extract(*target*, *out_format*)

class Raw(*instance=None*, *disk_name=None*, *path=None*)
Bases: nova.virt.libvirt.imagebackend.Image

correct_format()

create_image(*prepare_template*, *base*, *size*, **args*, ***kwargs*)

get_model(*connection*)

static is_file_in_instance_path()

snapshot_extract(*target*, *out_format*)

class Rbd(*instance=None*, *disk_name=None*, *path=None*, ***kwargs*)
Bases: nova.virt.libvirt.imagebackend.Image

SUPPORTS_CLONE = True

check_image_exists()

clone(*context*, *image_id_or_uri*)

create_image(*prepare_template*, *base*, *size*, **args*, ***kwargs*)

get_disk_size(*name*)
Returns the size of the virtual disk in bytes.

The name argument is ignored since this backend already knows its name, and callers may pass a non-existent local file path.

get_model(*connection*)

static is_shared_block_storage()

libvirt_info(*disk_bus*, *disk_dev*, *device_type*, *cache_mode*, *extra_specs*, *hypervisor_version*)
Get *LibvirtConfigGuestDisk* filled for this image.

Disk_dev Disk bus device name

Disk_bus Disk bus type

Device_type Device type for this image.

Cache_mode Caching mode for this image

Extra_specs Instance type extra specs dict.

snapshot_extract(*target*, *out_format*)

3.7.688 The nova.virt.libvirt.imagecache Module

Image cache manager.

The cache manager implements the specification at http://wiki.openstack.org/nova-image-cache-management.

class ImageCacheManager
Bases: nova.virt.imagecache.ImageCacheManager

update(*context*, *all_instances*)

get_cache_fname(*images*, *key*)
Return a filename based on the SHA1 hash of a given image ID.

Image files stored in the _base directory that match this pattern are considered for cleanup by the image cache manager. The cache manager considers the file to be in use if it matches an instance's image_ref, kernel_id or ramdisk_id property.

However, in grizzly-3 and before, only the image_ref property was considered. This means that it's unsafe to store kernel and ramdisk images using this pattern until we're sure that all compute nodes are running a cache manager newer than grizzly-3. For now, we require admins to confirm that by setting the remove_unused_kernels boolean but, at some point in the future, we'll be safely able to assume this.

get_info_filename(*base_path*)
Construct a filename for storing additional information about a base image.

Returns a filename.

is_valid_info_file(*path*)
Test if a given path matches the pattern for info files.

read_stored_checksum(*target*, *timestamped=True*)
Read the checksum.

Returns the checksum (as hex) or None.

read_stored_info(*target*, *field=None*, *timestamped=False*)
Read information about an image.

Returns an empty dictionary if there is no info, just the field value if a field is requested, or the entire dictionary otherwise.

write_stored_checksum(*target*)
Write a checksum to disk for a file in _base.

write_stored_info(*target*, *field=None*, *value=None*)
Write information about an image.

3.7.689 The nova.virt.libvirt.lvm Module

clear_volume(*path*)
Obfuscate the logical volume.

Parameters **path** – logical volume path

create_volume(*vg*, *lv*, *size*, *sparse=False*)
Create LVM image.

Creates a LVM image with given size.

Parameters

- **vg** – existing volume group which should hold this image
- **lv** – name for this image (logical volume)

Size size of image in bytes

Sparse create sparse logical volume

get_volume_group_info(*vg*)
Return free/used/total space info for a volume group in bytes

Parameters **vg** – volume group name

Returns A dict containing: :total: How big the filesystem is (in bytes) :free: How much space is free (in bytes) :used: How much space is used (in bytes)

get_volume_size(*path*)
Get logical volume size in bytes.

Parameters **path** – logical volume path

Raises processutils.ProcessExecutionError if getting the volume size fails in some unexpected way.

Raises exception.VolumeBDMPathNotFound if the volume path does not exist.

list_volumes(*vg*)
List logical volumes paths for given volume group.

Parameters **vg** – volume group name

Returns Return a logical volume list for given volume group : Data format example : ['volume-aaa', 'volume-bbb', 'volume-ccc']

remove_volumes(*paths*)
Remove one or more logical volume.

volume_info(*path*)
Get logical volume info.

Parameters **path** – logical volume path

Returns Return a dict object including info of given logical volume : Data format example : {'#Seg': '1', 'Move': '', 'Log': '', 'Meta%': '', 'Min': '-1', : ... : 'Free': '9983', 'LV': 'volume-aaa', 'Host': 'xyz.com', : 'Active': 'active', 'Path': '/dev/vg/volume-aaa', '#LV': '3', : 'Maj': '-1', 'VSize': '50.00g', 'VFree': '39.00g', 'Pool': '', : 'VG Tags': '', 'KMaj': '253', 'Convert': '', 'LProfile': '', : '#Ext': '12799', 'Attr': '-wi-a——', 'VG': 'vg', : ... : 'LSize': '1.00g', '#PV': '1', '#VMdaCps': 'unmanaged'}

3.7.690 The nova.virt.libvirt.quobyte Module

mount_volume(*volume*, *mnt_base*, *configfile=None*)
Wraps execute calls for mounting a Quobyte volume

umount_volume(*mnt_base*)
Wraps execute calls for unmouting a Quobyte volume

validate_volume(*mnt_base*)
Wraps execute calls for checking validity of a Quobyte volume

3.7.691 The nova.virt.libvirt.rbd_utils Module

class RADOSClient(*driver*, *pool=None*)
Bases: object

Context manager to simplify error handling for connecting to ceph.

class RBDDriver(*pool*, *ceph_conf*, *rbd_user*)
Bases: object

ceph_args()
List of command line parameters to be passed to ceph commands to reflect RBDDriver configuration such as RBD user name and location of ceph.conf.

cleanup_volumes(*instance*)

clone(*image_location*, *dest_name*)

exists(*name*, *pool=None*, *snapshot=None*)

get_mon_addrs()

get_pool_info()

import_image(*base*, *name*)
Import RBD volume from image file.

Uses the command line import instead of librbd since rbd import command detects zeroes to preserve sparseness in the image.

Base Path to image file

Name Name of RBD volume

is_cloneable(*image_location*, *image_meta*)

parse_url(*url*)

resize(*name*, *size*)
Resize RBD volume.

Name Name of RBD object

Size New size in bytes

size(*name*)

supports_layering()

class RBDVolumeProxy(*driver*, *name*, *pool=None*, *snapshot=None*, *read_only=False*)
Bases: object

Context manager for dealing with an existing rbd volume.

This handles connecting to rados and opening an ioctx automatically, and otherwise acts like a librbd Image object.

The underlying librados client and ioctx can be accessed as the attributes 'client' and 'ioctx'.

3.7.692 The nova.virt.libvirt.remotefs Module

mount_share(*mount_path*, *export_path*, *export_type*, *options=None*)
Mount a remote export to mount_path.

Parameters

- **mount_path** – place where the remote export will be mounted
- **export_path** – path of the export to be mounted

Export_type remote export type (e.g. cifs, nfs, etc.)

Options A list containing mount options

unmount_share(*mount_path*, *export_path*)
Unmount a remote share.

Parameters

- **mount_path** – remote export mount point
- **export_path** – path of the remote export to be unmounted

3.7.693 The nova.virt.libvirt.utils Module

chown(*path*, *owner*)
Change ownership of file or directory

Parameters

- **path** – File or directory whose ownership to change
- **owner** – Desired new owner (given as uid or username)

chown_for_id_maps(*path*, *id_maps*)
Change ownership of file or directory for an id mapped environment

Parameters

- **path** – File or directory whose ownership to change

- **id_maps** – List of type LibvirtConfigGuestIDMap

copy_image(*src*, *dest*, *host=None*, *receive=False*)
Copy a disk image to an existing directory

Parameters

- **src** – Source image
- **dest** – Destination path
- **host** – Remote host
- **receive** – Reverse the rsync direction

create_cow_image(*backing_file*, *path*, *size=None*)
Create COW image

Creates a COW image with the given backing file

Parameters

- **backing_file** – Existing image on which to base the COW image
- **path** – Desired location of the COW image

create_image(*disk_format*, *path*, *size*)
Create a disk image

Parameters

- **disk_format** – Disk image format (as known by qemu-img)
- **path** – Desired location of the disk image
- **size** – Desired size of disk image. May be given as an int or a string. If given as an int, it will be interpreted as bytes. If it's a string, it should consist of a number with an optional suffix ('K' for Kibibytes, M for Mebibytes, 'G' for Gibibytes, 'T' for Tebibytes). If no suffix is given, it will be interpreted as bytes.

execute(**args*, ***kwargs*)

extract_snapshot(*disk_path*, *source_fmt*, *out_path*, *dest_fmt*)
Extract a snapshot from a disk image. Note that nobody should write to the disk image during this operation.

Parameters

- **disk_path** – Path to disk image
- **out_path** – Desired path of extracted snapshot

fetch_image(*context*, *target*, *image_id*, *user_id*, *project_id*, *max_size=0*)
Grab image.

file_delete(*path*)
Delete (unlink) file

Note: The reason this is kept in a separate module is to easily be able to provide a stub module that doesn't alter system state at all (for unit tests)

file_open(**args*, ***kwargs*)
Open file

see built-in file() documentation for more details

Note: The reason this is kept in a separate module is to easily be able to provide a stub module that doesn't alter system state at all (for unit tests)

find_disk(*virt_dom*)
Find root device path for instance

May be file or device

get_arch(*image_meta*)
Determine the architecture of the guest (or host).

This method determines the CPU architecture that must be supported by the hypervisor. It gets the (guest) arch info from image_meta properties, and it will fallback to the nova-compute (host) arch if no architecture info is provided in image_meta.

Parameters **image_meta** – the metadata associated with the instance image

Returns guest (or host) architecture

get_disk_backing_file(*path*, *basename=True*)
Get the backing file of a disk image

Parameters **path** – Path to the disk image

Returns a path to the image's backing store

get_disk_size(*path*)
Get the (virtual) size of a disk image

Parameters **path** – Path to the disk image

Returns Size (in bytes) of the given disk image as it would be seen by a virtual machine.

get_disk_type(*path*)
Retrieve disk type (raw, qcow2, lvm) for given file.

get_fc_hbas()
Get the Fibre Channel HBA information.

get_fc_hbas_info()
Get Fibre Channel WWNs and device paths from the system, if any.

get_fc_wwnns()
Get Fibre Channel WWNNs from the system, if any.

get_fc_wwpns()
Get Fibre Channel WWPNs from the system, if any.

get_fs_info(*path*)
Get free/used/total space info for a filesystem

Parameters **path** – Any dirent on the filesystem

Returns

A dict containing:

free How much space is free (in bytes)

used How much space is used (in bytes)

total How big the filesystem is (in bytes)

get_instance_path(*instance*, *forceold=False*, *relative=False*)
Determine the correct path for instance storage.

This method determines the directory name for instance storage, while handling the fact that we changed the naming style to something more unique in the grizzly release.

Parameters

- **instance** – the instance we want a path for
- **forceold** – force the use of the pre-grizzly format
- **relative** – if True, just the relative path is returned

Returns a path to store information about that instance

get_instance_path_at_destination(*instance*, *migrate_data=None*)
Get the the instance path on destination node while live migration.

This method determines the directory name for instance storage on destination node, while live migration.

Parameters

- **instance** – the instance we want a path for
- **migrate_data** – if not None, it is a dict which holds data required for live migration without shared storage.

Returns a path to store information about that instance

get_iscsi_initiator()

is_mounted(*mount_path*, *source=None*)
Check if the given source is mounted at given destination point.

is_valid_hostname(*hostname*)

load_file(*path*)
Read contents of file

Parameters **path** – File to read

path_exists(*path*)
Returns if path exists

Note: **The reason this is kept in a separate module is to easily** be able to provide a stub module that doesn't alter system state at all (for unit tests)

perform_unit_add_for_s390(*device_number*, *target_wwn*, *lun*)
Write the LUN to the port's unit_add attribute.

perform_unit_remove_for_s390(*device_number*, *target_wwn*, *lun*)
Write the LUN to the port's unit_remove attribute.

pick_disk_driver_name(*hypervisor_version*, *is_block_dev=False*)
Pick the libvirt primary backend driver name

If the hypervisor supports multiple backend drivers we have to tell libvirt which one should be used.

Xen supports the following drivers: "tap", "tap2", "phy", "file", or "qemu", being "qemu" the preferred one. Qemu only supports "qemu".

Parameters **is_block_dev** –

Returns driver_name or None

write_to_file(*path*, *contents*, *umask=None*)
Write the given contents to a file

Parameters

- **path** – Destination file
- **contents** – Desired contents of the file
- **umask** – Umask to set when creating this file (will be reset)

3.7.694 The nova.virt.libvirt.vif Module

VIF drivers for libvirt.

class LibvirtGenericVIFDriver
Bases: object

Generic VIF driver for libvirt networking.

get_base_config(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_br_name(*iface_id*)

get_bridge_name(*vif*)

get_config(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_802qbg(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_802qbh(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_bridge(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)
Get VIF configurations for bridge type.

get_config_hw_veb(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_iovisor(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_ivs(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_ivs_ethernet(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_ivs_hybrid(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_midonet(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_mlnx_direct(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_ovs(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_ovs_bridge(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_ovs_hybrid(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_tap(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_vhostuser(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_config_vrouter(*instance*, *vif*, *image_meta*, *inst_type*, *virt_type*)

get_firewall_required(*vif*)

get_ovs_interfaceid(*vif*)

get_veth_pair_names(*iface_id*)

get_vif_devname(*vif*)

get_vif_devname_with_prefix(*vif*, *prefix*)

plug(*instance*, *vif*)

plug_802qbg(*instance*, *vif*)

plug_802qbh(*instance*, *vif*)

plug_bridge(*instance*, *vif*)
Ensure that the bridge exists, and add VIF to it.

plug_hw_veb(*instance*, *vif*)

plug_iovisor(*instance*, *vif*)
Plug using PLUMgrid IO Visor Driver

Connect a network device to their respective Virtual Domain in PLUMgrid Platform.

plug_ivs(*instance*, *vif*)

plug_ivs_ethernet(*instance*, *vif*)

plug_ivs_hybrid(*instance*, *vif*)
Plug using hybrid strategy (same as OVS)

Create a per-VIF linux bridge, then link that bridge to the OVS integration bridge via a veth device, setting up the other end of the veth device just like a normal IVS port. Then boot the VIF on the linux bridge using standard libvirt mechanisms.

plug_midonet(*instance*, *vif*)
Plug into MidoNet's network port

Bind the vif to a MidoNet virtual port.

plug_mlnx_direct(*instance*, *vif*)

plug_ovs(*instance*, *vif*)

plug_ovs_bridge(*instance*, *vif*)
No manual plugging required.

plug_ovs_hybrid(*instance*, *vif*)
Plug using hybrid strategy

Create a per-VIF linux bridge, then link that bridge to the OVS integration bridge via a veth device, setting up the other end of the veth device just like a normal OVS port. Then boot the VIF on the linux bridge using standard libvirt mechanisms.

plug_tap(*instance*, *vif*)
Plug a VIF_TYPE_TAP virtual interface.

plug_vhostuser(*instance*, *vif*)

plug_vrouter(*instance*, *vif*)
Plug into Contrail's network port

Bind the vif to a Contrail virtual port.

unplug(*instance*, *vif*)

unplug_802qbg(*instance*, *vif*)

unplug_802qbh(*instance*, *vif*)

unplug_bridge(*instance*, *vif*)
No manual unplugging required.

unplug_hw_veb(*instance*, *vif*)

unplug_iovisor(*instance*, *vif*)
Unplug using PLUMgrid IO Visor Driver

Delete network device and to their respective connection to the Virtual Domain in PLUMgrid Platform.

unplug_ivs(*instance*, *vif*)

unplug_ivs_ethernet(*instance*, *vif*)
Unplug the VIF by deleting the port from the bridge.

unplug_ivs_hybrid(*instance*, *vif*)
UnPlug using hybrid strategy (same as OVS)

Unhook port from IVS, unhook port from bridge, delete bridge, and delete both veth devices.

unplug_midonet(*instance*, *vif*)
Unplug from MidoNet network port

Unbind the vif from a MidoNet virtual port.

unplug_mlnx_direct(*instance*, *vif*)

unplug_ovs(*instance*, *vif*)

unplug_ovs_bridge(*instance*, *vif*)
No manual unplugging required.

unplug_ovs_hybrid(*instance*, *vif*)
UnPlug using hybrid strategy

Unhook port from OVS, unhook port from bridge, delete bridge, and delete both veth devices.

unplug_tap(*instance*, *vif*)
Unplug a VIF_TYPE_TAP virtual interface.

unplug_vhostuser(*instance*, *vif*)

unplug_vrouter(*instance*, *vif*)
Unplug Contrail's network port

Unbind the vif from a Contrail virtual port.

is_vif_model_valid_for_virt(*virt_type*, *vif_model*)

3.7.695 The nova.virt.libvirt.volume Module

Volume drivers for libvirt.

class LibvirtAOEVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Driver to attach AoE volumes to libvirt.

connect_volume(*connection_info*, *mount_device*)

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtBaseVolumeDriver(*connection*, *is_block_dev*)
Bases: object

Base class for volume drivers.

connect_volume(*connection_info*, *disk_info*)
Connect the volume. Returns xml for libvirt.

disconnect_volume(*connection_info*, *disk_dev*)
Disconnect the volume.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtFakeVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Driver to attach fake volumes to libvirt.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtFibreChannelVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Driver to attach Fibre Channel Network volumes to libvirt.

connect_volume(**args*, ***kwargs*)
Attach the volume to instance_name.

disconnect_volume(**args*, ***kwargs*)
Detach the volume from instance_name.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtGPFSVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Class for volumes backed by gpfs volume.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtGlusterfsVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Class implements libvirt part of volume driver for GlusterFS.

connect_volume(*connection_info*, *mount_device*)

disconnect_volume(*connection_info*, *disk_dev*)
Disconnect the volume.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtISCSIVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Driver to attach Network volumes to libvirt.

connect_volume(**args*, ***kwargs*)
Attach the volume to instance_name.

disconnect_volume(**args*, ***kwargs*)
Detach the volume from instance_name.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

supported_transports = ['be2iscsi', 'bnx2i', 'cxgb3i', 'cxgb4i', 'qla4xxx', 'ocs']

class LibvirtISERVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtISCSIVolumeDriver

Driver to attach Network volumes to libvirt.

class LibvirtNFSVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Class implements libvirt part of volume driver for NFS.

connect_volume(*connection_info*, *disk_info*)
Connect the volume. Returns xml for libvirt.

disconnect_volume(*connection_info*, *disk_dev*)
Disconnect the volume.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtNetVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Driver to attach Network volumes to libvirt.

disconnect_volume(*connection_info*, *disk_dev*)
Detach the volume from instance_name.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtQuobyteVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Class implements libvirt part of volume driver for Quobyte.

connect_volume(**args*, ***kwargs*)
Connect the volume.

disconnect_volume(**args*, ***kwargs*)
Disconnect the volume.

get_config(*connection_info*, *disk_info*)

class LibvirtSMBFSVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Class implements libvirt part of volume driver for SMBFS.

connect_volume(*connection_info*, *disk_info*)
Connect the volume.

disconnect_volume(*connection_info*, *disk_dev*)
Disconnect the volume.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtScalityVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Scality SOFS Nova driver. Provide hypervisors with access to sparse files on SOFS.

connect_volume(*connection_info*, *disk_info*)
Connect the volume. Returns xml for libvirt.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

class LibvirtVolumeDriver(*connection*)
Bases: nova.virt.libvirt.volume.LibvirtBaseVolumeDriver

Class for volumes backed by local file.

get_config(*connection_info*, *disk_info*)
Returns xml for libvirt.

3.7.696 The nova.virt.netutils Module

Network-related utilities for supporting libvirt connection code.

get_injected_network_template(*network_info*, *use_ipv6=None*, *template=None*, *libvirt_virt_type=None*)
Returns a rendered network template for the given network_info.

Parameters

- **network_info** – `get_instance_nw_info()`
- **use_ipv6** – If False, do not return IPv6 template information even if an IPv6 subnet is present in network_info.
- **template** – Path to the interfaces template file.
- **libvirt_virt_type** – The Libvirt *virt_type*, will be *None* for other hypervisors..

get_ip_version(*cidr*)

get_net_and_mask(*cidr*)

get_net_and_prefixlen(*cidr*)

get_network_metadata(*network_info*, *use_ipv6=None*)
Gets a more complete representation of the instance network information.

This data is exposed as network_data.json in the metadata service and the config drive.

Parameters

- **network_info** – *nova.network.models.NetworkInfo* object describing the network metadata.
- **use_ipv6** – If False, do not return IPv6 template information even if an IPv6 subnet is present in network_info. Defaults to nova.netconf.use_ipv6.

3.7.697 The nova.virt.opts Module

list_opts()

3.7.698 The nova.virt.storage_users Module

get_storage_users(*storage_path*)
Get a list of all the users of this storage path.

register_storage_use(*storage_path*, *hostname*)
Identify the id of this instance storage.

3.7.699 The nova.virt.virtapi Module

class VirtAPI
Bases: object

provider_fw_rule_get_all(*context*)
Get the provider firewall rules :param context: security context

wait_for_instance_event(**args*, ***kwds*)

3.7.700 The nova.virt.vmwareapi.constants Module

Shared constants across the VMware driver

3.7.701 The nova.virt.vmwareapi.driver Module

A connection to the VMware vCenter platform.

class VMwareAPISession(*host_ip=None*, *host_port=443*, *username=None*, *password=None*, *retry_count=10*, *scheme='https'*, *cacert=None*, *insecure=False*)
Bases: oslo_vmware.api.VMwareAPISession

Sets up a session with the VC/ESX host and handles all the calls made to the host.

class VMwareVCDriver(*virtapi*, *scheme='https'*)
Bases: nova.virt.driver.ComputeDriver

The VC host connection object.

LEGACY_NODENAME = <_sre.SRE_Pattern object at 0x7f857357f108>

attach_interface(*instance*, *image_meta*, *vif*)
Attach an interface to the instance.

attach_volume(*context*, *connection_info*, *instance*, *mountpoint*, *disk_bus=None*, *device_type=None*, *encryption=None*)
Attach volume storage to VM instance.

capabilities = {'supports_recreate': False, 'has_imagecache': True, 'supports_migrate_to_same_host': True}

cleanup(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*, *destroy_vifs=True*)
Cleanup after instance being destroyed by Hypervisor.

cleanup_host(*host*)

confirm_migration(*migration*, *instance*, *network_info*)
Confirms a resize, destroying the source VM.

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)
Destroy VM instance.

detach_interface(*instance*, *vif*)
Detach an interface from the instance.

detach_volume(*connection_info*, *instance*, *mountpoint*, *encryption=None*)
Detach volume storage to VM instance.

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)
Completes a resize, turning on the migrated instance.

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)
Finish reverting a resize, powering back on the instance.

get_available_nodes(*refresh=False*)
Returns nodenames of all nodes managed by the compute service.

This method is for multi compute-nodes support. If a driver supports multi compute-nodes, this method returns a list of nodenames managed by the service. Otherwise, this method should return [hypervisor_hostname].

get_available_resource(*nodename*)
Retrieve resource info.

This method is called when nova-compute launches, and as part of a periodic task.

Returns dictionary describing resources

get_diagnostics(*instance*)
Return data about VM diagnostics.

get_host_ip_addr()
Returns the IP address of the vCenter host.

get_host_uptime()
Host uptime operation not supported by VC driver.

get_info(*instance*)
Return info about the VM instance.

get_instance_diagnostics(*instance*)
Return data about VM diagnostics.

get_instance_disk_info(*instance*, *block_device_info=None*)

get_vnc_console(*context*, *instance*)
Return link to instance's VNC console using vCenter logic.

get_volume_connector(*instance*)
Return volume connector information.

host_maintenance_mode(*host*, *mode*)
Host operations not supported by VC driver.

This needs to override the ESX driver implementation.

host_power_action(*action*)
Host operations not supported by VC driver.

This needs to override the ESX driver implementation.

init_host(*host*)

inject_network_info(*instance*, *nw_info*)
inject network info for specified instance.

instance_exists(*instance*)
Efficient override of base instance_exists method.

list_instance_uuids()
List VM instance UUIDs.

list_instances()
List VM instances from all nodes.

live_migration(*context*, *instance*, *dest*, *post_method*, *recover_method*, *block_migration=False*, *migrate_data=None*)
Live migration of an instance to another host.

manage_image_cache(*context*, *all_instances*)
Manage the local cache of images.

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)
Transfers the disk of a running instance in multiple phases, turning off the instance before the end.

need_legacy_block_device_info

pause(*instance*)
Pause VM instance.

poll_rebooting_instances(*timeout*, *instances*)
Poll for rebooting instances.

power_off(*instance*, *timeout=0*, *retry_interval=0*)
Power off the specified instance.

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)
Power on the specified instance.

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)
Reboot VM instance.

rescue(*context*, *instance*, *network_info*, *image_meta*, *rescue_password*)
Rescue the specified instance.

resume(*context*, *instance*, *network_info*, *block_device_info=None*)
Resume the suspended VM instance.

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)
resume guest state when a host is booted.

rollback_live_migration_at_destination(*context*, *instance*, *network_info*, *block_device_info*, *destroy_disks=True*, *migrate_data=None*)
Clean up destination node after a failed live migration.

set_host_enabled(*enabled*)
Host operations not supported by VC driver.

This needs to override the ESX driver implementation.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Create snapshot from a running VM instance.

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)
Create VM instance.

suspend(*context*, *instance*)
Suspend the specified instance.

unpause(*instance*)
Unpause paused VM instance.

unrescue(*instance*, *network_info*)
Unrescue the specified instance.

3.7.702 The nova.virt.vmwareapi.ds_util Module

Datastore utility functions

disk_copy(*session*, *dc_ref*, *src_file*, *dst_file*)
Copies the source virtual disk to the destination.

disk_delete(*session*, *dc_ref*, *file_path*)
Deletes a virtual disk.

disk_move(*session*, *dc_ref*, *src_file*, *dst_file*)
Moves the source virtual disk to the destination.

The list of possible faults that the server can return on error include:

•CannotAccessFile: Thrown if the source file or folder cannot be moved because of insufficient permissions.

•FileAlreadyExists: Thrown if a file with the given name already exists at the destination.

•FileFault: Thrown if there is a generic file error

•FileLocked: Thrown if the source file or folder is currently locked or in use.

•FileNotFound: Thrown if the file or folder specified by sourceName is not found.

•InvalidDatastore: Thrown if the operation cannot be performed on the source or destination datastores.

•NoDiskSpace: Thrown if there is not enough space available on the destination datastore.

•RuntimeFault: Thrown if any type of runtime fault is thrown that is not covered by the other faults; for example, a communication error.

file_copy(*session*, *src_file*, *src_dc_ref*, *dst_file*, *dst_dc_ref*)

file_delete(*session*, *ds_path*, *dc_ref*)

file_exists(*session*, *ds_browser*, *ds_path*, *file_name*)
Check if the file exists on the datastore.

file_move(*session*, *dc_ref*, *src_file*, *dst_file*)
Moves the source file or folder to the destination.

The list of possible faults that the server can return on error include:

•CannotAccessFile: Thrown if the source file or folder cannot be moved because of insufficient permissions.

•FileAlreadyExists: Thrown if a file with the given name already exists at the destination.

•FileFault: Thrown if there is a generic file error

•FileLocked: Thrown if the source file or folder is currently locked or in use.

•FileNotFound: Thrown if the file or folder specified by sourceName is not found.

•InvalidDatastore: Thrown if the operation cannot be performed on the source or destination datastores.

•NoDiskSpace: Thrown if there is not enough space available on the destination datastore.

•RuntimeFault: Thrown if any type of runtime fault is thrown that is not covered by the other faults; for example, a communication error.

get_allowed_datastore_types(*disk_type*)

get_available_datastores(*session*, *cluster=None*, *datastore_regex=None*)
Get the datastore list and choose the first local storage.

get_datastore(*session*, *cluster*, *datastore_regex=None*, *storage_policy=None*, *allowed_ds_types=frozenset(['vsan', 'NFS', 'VMFS', 'NFS41'])*)
Get the datastore list and choose the most preferable one.

get_sub_folders(*session*, *ds_browser*, *ds_path*)
Return a set of subfolders for a path on a datastore.

If the path does not exist then an empty set is returned.

mkdir(*session*, *ds_path*, *dc_ref*)
Creates a directory at the path specified. If it is just "NAME", then a directory with this name is created at the topmost level of the DataStore.

search_datastore_spec(*client_factory*, *file_name*)
Builds the datastore search spec.

3.7.703 The nova.virt.vmwareapi.error_util Module

Exception classes specific for the VMware driver.

exception NoRootDiskDefined(*message=None*, *details=None*, ***kwargs*)
Bases: oslo_vmware.exceptions.VMwareDriverException

msg_fmt = u'No root disk defined.'

exception PbmDefaultPolicyDoesNotExist(*message=None*, *details=None*, ***kwargs*)
Bases: oslo_vmware.exceptions.VMwareDriverConfigurationException

msg_fmt = u"The default PBM policy doesn't exist on the backend."

exception PbmDefaultPolicyUnspecified(*message=None*, *details=None*, ***kwargs*)
Bases: oslo_vmware.exceptions.VMwareDriverConfigurationException

msg_fmt = u'Default PBM policy is required if PBM is enabled.'

3.7.704 The nova.virt.vmwareapi.host Module

Management class for host-related functions (start, reboot, etc).

class VCState(*session*, *host_name*, *cluster*, *datastore_regex*)
Bases: object

Manages information about the VC host this compute node is running on.

get_host_stats(*refresh=False*)
Return the current state of the host. If 'refresh' is True, run the update first.

update_status()
Update the current state of the cluster.

3.7.705 The nova.virt.vmwareapi.imagecache Module

Image cache class

Images that are stored in the cache folder will be stored in a folder whose name is the image ID. In the event that an image is discovered to be no longer used then a timestamp will be added to the image folder. The timestamp will be a folder - this is due to the fact that we can use the VMware API's for creating and deleting of folders (it really simplifies things). The timestamp will contain the time, on the compute node, when the image was first seen to be unused. At each aging iteration we check if the image can be aged. This is done by comparing the current nova compute time to

the time embedded in the timestamp. If the time exceeds the configured aging time then the parent folder, that is the image ID folder, will be deleted. That effectively ages the cached image. If an image is used then the timestamps will be deleted.

When accessing a timestamp we make use of locking. This ensure that aging will not delete an image during the spawn operation. When spawning the timestamp folder will be locked and the timestamps will be purged. This will ensure that a image is not deleted during the spawn.

class ImageCacheManager(*session*, *base_folder*)
Bases: nova.virt.imagecache.ImageCacheManager

enlist_image(*image_id*, *datastore*, *dc_ref*)

get_image_cache_folder(*datastore*, *image_id*)
Returns datastore path of folder containing the image.

timestamp_cleanup(*dc_ref*, *ds_browser*, *ds_path*)

timestamp_folder_get(*ds_path*, *image_id*)
Returns the timestamp folder.

update(*context*, *instances*, *datastores_info*)
The cache manager entry point.

This will invoke the cache manager. This will update the cache according to the defined cache management scheme. The information populated in the cached stats will be used for the cache management.

3.7.706 The nova.virt.vmwareapi.images Module

Utility functions for Image transfer and manipulation.

class VMwareImage(*image_id*, *file_size=0*, *os_type='otherGuest'*, *adapter_type='lsiLogic'*, *disk_type='preallocated'*, *container_format='bare'*, *file_type='vmdk'*, *linked_clone=None*, *vif_model='e1000'*)
Bases: object

file_size_in_kb

classmethod from_image(*image_id*, *image_meta*)
Returns VMwareImage, the subset of properties the driver uses.

:param image_id - image id of image :param image_meta - image metadata object we are working with :return: vmware image object :rtype: nova.virt.vmwareapi.images.VmwareImage

is_iso

is_ova

is_sparse

fetch_image(*context*, *instance*, *host*, *port*, *dc_name*, *ds_name*, *file_path*, *cookies=None*)
Download image from the glance image server.

fetch_image_ova(*context*, *instance*, *session*, *vm_name*, *ds_name*, *vm_folder_ref*, *res_pool_ref*)
Download the OVA image from the glance image server to the Nova compute node.

fetch_image_stream_optimized(*context*, *instance*, *session*, *vm_name*, *ds_name*, *vm_folder_ref*, *res_pool_ref*)
Fetch image from Glance to ESX datastore.

get_vmdk_name_from_ovf(*xmlstr*)
Parse the OVA descriptor to extract the vmdk name.

start_transfer(*context*, *read_file_handle*, *data_size*, *write_file_handle=None*, *image_id=None*, *image_meta=None*)
Start the data transfer from the reader to the writer. Reader writes to the pipe and the writer reads from the pipe. This means that the total transfer time boils down to the slower of the read/write and not the addition of the two times.

upload_image_stream_optimized(*context*, *image_id*, *instance*, *session*, *vm*, *vmdk_size*)
Upload the snapshotted vm disk file to Glance image server.

upload_iso_to_datastore(*iso_path*, *instance*, ***kwargs*)

3.7.707 The nova.virt.vmwareapi.io_util Module

Utility classes for defining the time saving transfer of data from the reader to the write using a LightQueue as a Pipe between the reader and the writer.

class GlanceWriteThread(*context*, *input*, *image_id*, *image_meta=None*)
Bases: object

Ensures that image data is written to in the glance client and that it is in correct ('active')state.

close()

start()

stop()

wait()

class IOThread(*input*, *output*)
Bases: object

Class that reads chunks from the input file and writes them to the output file till the transfer is completely done.

start()

stop()

wait()

class ThreadSafePipe(*maxsize*, *transfer_size*)
Bases: eventlet.queue.LightQueue

The pipe to hold the data which the reader writes to and the writer reads from.

close()
A place-holder to maintain consistency.

read(*chunk_size*)
Read data from the pipe.

Chunksize if ignored for we have ensured that the data chunks written to the pipe by readers is the same as the chunks asked for by the Writer.

seek(*offset*, *whence=0*)
Set the file's current position at the offset.

tell()
Get size of the file to be read.

write(*data*)
Put a data item in the pipe.

3.7.708 The nova.virt.vmwareapi.network_util Module

Utility functions for ESX Networking.

check_if_vlan_interface_exists(*session*, *vlan_interface*, *cluster=None*)
Checks if the vlan_interface exists on the esx host.

create_port_group(*session*, *pg_name*, *vswitch_name*, *vlan_id=0*, *cluster=None*)
Creates a port group on the host system with the vlan tags supplied. VLAN id 0 means no vlan id association.

get_network_with_the_name(*session*, *network_name='vmnet0'*, *cluster=None*)
Gets reference to the network whose name is passed as the argument.

get_vlanid_and_vswitch_for_portgroup(*session*, *pg_name*, *cluster=None*)
Get the vlan id and vswitch associated with the port group.

get_vswitch_for_vlan_interface(*session*, *vlan_interface*, *cluster=None*)
Gets the vswitch associated with the physical network adapter with the name supplied.

3.7.709 The nova.virt.vmwareapi.read_write_util Module

Classes to handle image files

Collection of classes to handle image upload/download to/from Image service (like Glance image storage and retrieval service) from/to ESX/ESXi server.

class VMwareHTTPReadFile(*host*, *port*, *data_center_name*, *datastore_name*, *cookies*, *file_path*, *scheme='https'*)
Bases: `oslo_vmware.rw_handles.FileHandle`

VMware file read handler class.

get_size()
Get size of the file to be read.

read(*chunk_size*)

3.7.710 The nova.virt.vmwareapi.vif Module

VIF drivers for VMware.

ensure_vlan_bridge(*session*, *vif*, *cluster=None*, *create_vlan=True*)
Create a vlan and bridge unless they already exist.

get_network_device(*hardware_devices*, *mac_address*)
Return the network device with MAC 'mac_address'.

get_network_ref(*session*, *cluster*, *vif*, *is_neutron*)

get_neutron_network(*session*, *network_name*, *cluster*, *vif*)

get_vif_dict(*session*, *cluster*, *vif_model*, *is_neutron*, *vif*)

get_vif_info(*session*, *cluster*, *is_neutron*, *vif_model*, *network_info*)

3.7.711 The nova.virt.vmwareapi.vim_util Module

The VMware API utility module.

cancel_retrieve(*vim*, *token*)
Cancels the retrieve operation.

get_about_info(*vim*)
Get the About Info from the service content.

get_dynamic_properties(*vim*, *mobj*, *type*, *property_names*)
Gets the specified properties of the Managed Object.

get_dynamic_property(*vim*, *mobj*, *type*, *property_name*)
Gets a particular property of the Managed Object.

get_inner_objects(*vim*, *base_obj*, *path*, *inner_type*, *properties_to_collect=None*, *all=False*)
Gets the list of inner objects of the type specified.

get_obj_spec(*client_factory*, *obj*, *select_set=None*)
Builds the Object Spec object.

get_object_properties(*vim*, *collector*, *mobj*, *type*, *properties*)
Gets the properties of the Managed object specified.

get_objects(*vim*, *type*, *properties_to_collect=None*, *all=False*)
Gets the list of objects of the type specified.

get_prop_filter_spec(*client_factory*, *obj_spec*, *prop_spec*)
Builds the Property Filter Spec Object.

get_prop_spec(*client_factory*, *spec_type*, *properties*)
Builds the Property Spec Object.

get_properties_for_a_collection_of_objects(*vim*, *type*, *obj_list*, *properties*)
Gets the list of properties for the collection of objects of the type specified.

object_to_dict(*obj*, *list_depth=1*)
Convert Suds object into serializable format.

The calling function can limit the amount of list entries that are converted.

3.7.712 The nova.virt.vmwareapi.vm_util Module

The VMware API VM utility module to build SOAP object specs.

class CpuLimits(*cpu_limit=None*, *cpu_reservation=None*, *cpu_shares_level=None*, *cpu_shares_share=None*)
Bases: object

validate()

class ExtraSpecs(*cpu_limits=None*, *hw_version=None*, *storage_policy=None*)
Bases: object

has_cpu_limits()

class VmdkInfo
Bases: tuple

VmdkInfo(path, adapter_type, disk_type, capacity_in_bytes, device)

adapter_type
Alias for field number 1

capacity_in_bytes
Alias for field number 3

device
Alias for field number 4

disk_type
Alias for field number 2

path
Alias for field number 0

allocate_controller_key_and_unit_number(*client_factory*, *devices*, *adapter_type*)
This function inspects the current set of hardware devices and returns controller_key and unit_number that can be used for attaching a new virtual disk to adapter with the given adapter_type.

clone_vm_spec(*client_factory*, *location*, *power_on=False*, *snapshot=None*, *template=False*, *config=None*)
Builds the VM clone spec.

convert_vif_model(*name*)
Converts standard VIF_MODEL types to the internal VMware ones.

copy_virtual_disk(*session*, *dc_ref*, *source*, *dest*)
Copy a sparse virtual disk to a thin virtual disk.

This is also done to generate the meta-data file whose specifics depend on the size of the disk, thin/thick provisioning and the storage adapter type.

Parameters

- **session** –
  - session for connection
- **dc_ref** –
  - data center reference object
- **source** –
  - source datastore path
- **dest** –
  - destination datastore path

Returns None

create_controller_spec(*client_factory*, *key*, *adapter_type='lsiLogic'*)
Builds a Config Spec for the LSI or Bus Logic Controller's addition which acts as the controller for the virtual hard disk to be attached to the VM.

create_virtual_cdrom_spec(*client_factory*, *datastore*, *controller_key*, *file_path*, *cdrom_unit_number*)
Builds spec for the creation of a new Virtual CDROM to the VM.

create_virtual_disk(*session*, *dc_ref*, *adapter_type*, *disk_type*, *virtual_disk_path*, *size_in_kb*)

create_virtual_disk_spec(*client_factory*, *controller_key*, *disk_type='preallocated'*, *file_path=None*, *disk_size=None*, *linked_clone=False*, *unit_number=None*, *device_name=None*)
Builds spec for the creation of a new/ attaching of an already existing Virtual Disk to the VM.

create_vm(*session*, *instance*, *vm_folder*, *config_spec*, *res_pool_ref*)
Create VM on ESX host.

destroy_vm(*session*, *instance*, *vm_ref=None*)
Destroy a VM instance. Assumes VM is powered off.

detach_devices_from_vm(*session*, *vm_ref*, *devices*)
Detach specified devices from VM.

detach_virtual_disk_spec(*client_factory*, *device*, *destroy_disk=False*)
Builds spec for the detach of an already existing Virtual Disk from VM.

find_entity_mor(*entity_list*, *entity_name*)
Returns managed object ref for given cluster or resource pool name.

find_rescue_device(*hardware_devices*, *instance*)
Returns the rescue device.

The method will raise an exception if the rescue device does not exist. The resuce device has suffix '-rescue.vmdk'. :param hardware_devices: the hardware devices for the instance :param instance: nova.objects.instance.Instance object :return: the rescue disk device object

get_add_vswitch_port_group_spec(*client_factory*, *vswitch_name*, *port_group_name*, *vlan_id*)
Builds the virtual switch port group add spec.

get_all_cluster_mors(*session*)
Get all the clusters in the vCenter.

get_all_cluster_refs_by_name(*session*, *path_list*)
Get reference to the Cluster, ResourcePool with the path specified.

The path is the display name. This can be the full path as well. The input will have the list of clusters and resource pool names

get_all_res_pool_mors(*session*)
Get all the resource pools in the vCenter.

get_attach_port_index(*session*, *vm_ref*)
Get the first free port index.

get_cdrom_attach_config_spec(*client_factory*, *datastore*, *file_path*, *controller_key*, *cdrom_unit_number*)
Builds and returns the cdrom attach config spec.

get_dict_mor(*session*, *list_obj*)
The input is a list of objects in the form (manage_object,display_name) The managed object will be in the form { value = "domain-1002", _type = "ClusterComputeResource" }

Output data format: | dict_mors = { | 'respool-1001': { 'cluster_mor': clusterMor, | 'res_pool_mor': resourcePoolMor, | 'name': display_name }, | 'domain-1002': { 'cluster_mor': clusterMor, | 'res_pool_mor': resourcePoolMor, | 'name': display_name }, | }

get_dynamic_property_mor(*session*, *mor_ref*, *attribute*)
Get the value of an attribute for a given managed object.

get_ephemeral_name(*id*)

get_ephemerals(*session*, *vm_ref*)

get_host_name_for_vm(*session*, *instance*)
Get the hostname of the ESXi host currently running an instance.

get_host_ref(*session*, *cluster=None*)
Get reference to a host within the cluster specified.

get_host_ref_for_vm(*session*, *instance*)
Get a MoRef to the ESXi host currently running an instance.

get_machine_id_change_spec(*client_factory*, *machine_id_str*)
Builds the machine id change config spec.

get_network_attach_config_spec(*client_factory*, *vif_info*, *index*)
Builds the vif attach config spec.

get_network_detach_config_spec(*client_factory*, *device*, *port_index*)
Builds the vif detach config spec.

get_rdm_disk(*hardware_devices*, *uuid*)
Gets the RDM disk key.

get_res_pool_ref(*session*, *cluster*)
Get the resource pool.

get_scsi_adapter_type(*hardware_devices*)
Selects a proper iscsi adapter type from the existing hardware devices

get_stats_from_cluster(*session*, *cluster*)
Get the aggregate resource stats of a cluster.

get_storage_profile_spec(*session*, *storage_policy*)
Gets the vm profile spec configured for storage policy.

get_values_from_object_properties(*session*, *props*)
Get the specific values from a object list.

The object values will be returned as a dictionary.

get_vm_boot_spec(*client_factory*, *device*)
Returns updated boot settings for the instance.

The boot order for the instance will be changed to have the input device as the boot disk.

get_vm_create_spec(*client_factory*, *instance*, *data_store_name*, *vif_infos*, *extra_specs*, *os_type='otherGuest'*, *profile_spec=None*, *metadata=None*)
Builds the VM Create spec.

get_vm_detach_port_index(*session*, *vm_ref*, *iface_id*)

get_vm_extra_config_spec(*client_factory*, *extra_opts*)
Builds extra spec fields from a dictionary.

get_vm_ref(*session*, *instance*)
Get reference to the VM through uuid or vm name.

get_vm_ref_from_name(*session*, *name*)

get_vm_resize_spec(*client_factory*, *vcpus*, *memory_mb*, *extra_specs*, *metadata=None*)
Provides updates for a VM spec.

get_vm_state(*session*, *instance*)

get_vmdk_adapter_type(*adapter_type*)
Return the adapter type to be used in vmdk descriptor.

Adapter type in vmdk descriptor is same for LSI-SAS, LSILogic & ParaVirtual because Virtual Disk Manager API does not recognize the newer controller types.

get_vmdk_attach_config_spec(*client_factory*, *disk_type='preallocated'*, *file_path=None*, *disk_size=None*, *linked_clone=False*, *controller_key=None*, *unit_number=None*, *device_name=None*)
Builds the vmdk attach config spec.

get_vmdk_backed_disk_device(*hardware_devices*, *uuid*)

get_vmdk_create_spec(*client_factory*, *size_in_kb*, *adapter_type='lsiLogic'*, *disk_type='preallocated'*)
Builds the virtual disk create spec.

get_vmdk_detach_config_spec(*client_factory*, *device*, *destroy_disk=False*)
Builds the vmdk detach config spec.

get_vmdk_info(*session*, *vm_ref*, *uuid=None*)
Returns information for the primary VMDK attached to the given VM.

get_vmdk_volume_disk(*hardware_devices*, *path=None*)

get_vnc_config_spec(*client_factory*, *port*)
Builds the vnc config spec.

get_vnc_port(*session*)
Return VNC port for an VM or None if there is no available port.

power_off_instance(*session*, *instance*, *vm_ref=None*)
Power off the specified instance.

power_on_instance(*session*, *instance*, *vm_ref=None*)
Power on the specified instance.

propset_dict(*propset*)
Turn a propset list into a dictionary

PropSet is an optional attribute on ObjectContent objects that are returned by the VMware API.

You can read more about these at: | http://pubs.vmware.com/vsphere-51/index.jsp | #com.vmware.wssdk.apiref.doc/ | vmodl.query.PropertyCollector.ObjectContent.html

Parameters **propset** – a property "set" from ObjectContent

Returns dictionary representing property set

reconfigure_vm(*session*, *vm_ref*, *config_spec*)
Reconfigure a VM according to the config spec.

relocate_vm_spec(*client_factory*, *datastore=None*, *host=None*, *disk_move_type='moveAllDiskBackingsAndAllowSharing'*)
Builds the VM relocation spec.

search_vm_ref_by_identifier(*session*, *identifier*)
Searches VM reference using the identifier.

This method is primarily meant to separate out part of the logic for vm_ref search that could be use directly in the special case of migrating the instance. For querying VM linked to an instance always use get_vm_ref instead.

vm_ref_cache_delete(*id*)

vm_ref_cache_from_instance(*func*)

vm_ref_cache_from_name(*func*)

vm_ref_cache_get(*id*)

vm_ref_cache_update(*id*, *vm_ref*)

vm_refs_cache_reset()

3.7.713 The nova.virt.vmwareapi.vmops Module

Class for VM tasks like spawn, snapshot, suspend, resume etc.

class DcInfo
Bases: tuple

DcInfo(ref, name, vmFolder)

name
Alias for field number 1

ref
Alias for field number 0

vmFolder
Alias for field number 2

class VMwareVMOps(*session*, *virtapi*, *volumeops*, *cluster=None*, *datastore_regex=None*)
Bases: `object`

Management class for VM-related tasks.

attach_interface(*instance*, *image_meta*, *vif*)
Attach an interface to the instance.

build_virtual_machine(*instance*, *image_info*, *dc_info*, *datastore*, *network_info*, *extra_specs*, *metadata*)

check_cache_folder(*ds_name*, *ds_ref*)
Check that the cache folder exists.

check_temp_folder(*ds_name*, *ds_ref*)
Check that the temp folder exists.

confirm_migration(*migration*, *instance*, *network_info*)
Confirms a resize, destroying the source VM.

destroy(*instance*, *destroy_disks=True*)
Destroy a VM instance.

Steps followed for each VM are: 1. Power off, if it is in poweredOn state. 2. Un-register. 3. Delete the contents of the folder holding the VM related data.

detach_interface(*instance*, *vif*)
Detach an interface from the instance.

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance=False*, *block_device_info=None*, *power_on=True*)
Completes a resize, turning on the migrated instance.

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info*, *power_on=True*)
Finish reverting a resize.

get_datacenter_ref_and_name(*ds_ref*)
Get the datacenter name and the reference.

get_diagnostics(*instance*)
Return data about VM diagnostics.

get_info(*instance*)
Return data about the VM instance.

get_instance_diagnostics(*instance*)
Return data about VM diagnostics.

get_vnc_console(*instance*)
Return connection info for a vnc console using vCenter logic.

inject_network_info(*instance*, *network_info*)
inject network info for specified instance.

instance_exists(*instance*)

list_instances()
Lists the VM instances that are registered with vCenter cluster.

live_migration(*context*, *instance_ref*, *dest*, *post_method*, *recover_method*, *block_migration=False*)
Spawning live_migration operation for distributing high-load.

manage_image_cache(*context*, *instances*)

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*)
Transfers the disk of a running instance in multiple phases, turning off the instance before the end.

pause(*instance*)

poll_rebooting_instances(*timeout*, *instances*)
Poll for rebooting instances.

power_off(*instance*)
Power off the specified instance.

Parameters **instance** – nova.objects.instance.Instance

power_on(*instance*)

reboot(*instance*, *network_info*, *reboot_type='SOFT'*)
Reboot a VM instance.

rescue(*context*, *instance*, *network_info*, *image_meta*)
Rescue the specified instance.

Attach the image that the instance was created from and boot from it.

resume(*instance*)
Resume the specified instance.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Create snapshot from a running VM instance.

Steps followed are:

1. Get the name of the vmdk file which the VM points to right now. Can be a chain of snapshots, so we need to know the last in the chain.
2. Create the snapshot. A new vmdk is created which the VM points to now. The earlier vmdk becomes read-only.
3. Creates a linked clone VM from the snapshot
4. Exports the disk in the link clone VM as a streamOptimized disk.
5. Delete the linked clone VM
6. Deletes the snapshot in original instance.

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info*, *block_device_info=None*)

suspend(*instance*)
Suspend the specified instance.

unpause(*instance*)

unrescue(*instance*, *power_on=True*)
Unrescue the specified instance.

class **VirtualMachineInstanceConfigInfo**(*instance*, *image_info*, *datastore*, *dc_info*, *image_cache*)
Bases: object

Parameters needed to create and configure a new instance.

cache_image_folder

cache_image_path

retry_if_task_in_progress(*f*)

3.7.714 The nova.virt.vmwareapi.volumeops Module

Management class for Storage-related functions (attach, detach, etc).

class **VMwareVolumeOps**(*session*, *cluster=None*)
Bases: object

Management class for Volume-related tasks.

attach_disk_to_vm(*vm_ref*, *instance*, *adapter_type*, *disk_type*, *vmdk_path=None*, *disk_size=None*, *linked_clone=False*, *device_name=None*)
Attach disk to VM by reconfiguration.

attach_root_volume(*connection_info*, *instance*, *datastore*, *adapter_type=None*)
Attach a root volume to the VM instance.

attach_volume(*connection_info*, *instance*, *adapter_type=None*)
Attach volume storage to VM instance.

detach_disk_from_vm(*vm_ref*, *instance*, *device*, *destroy_disk=False*)
Detach disk from VM by reconfiguration.

detach_volume(*connection_info*, *instance*)
Detach volume storage to VM instance.

get_volume_connector(*instance*)
Return volume connector information.

3.7.715 The nova.virt.volumeutils Module

Volume utilities for virt drivers.

get_iscsi_initiator()
Get iscsi initiator name for this machine.

3.7.716 The nova.virt.watchdog_actions Module

Describes and verifies the watchdog device actions.

is_valid_watchdog_action(*val*)
Check if the given value is a valid watchdog device parameter.

3.7.717 The nova.virt.xenapi.agent Module

class **SimpleDH**
Bases: object

This class wraps all the functionality needed to implement basic Diffie-Hellman-Merkle key exchange in Python. It features intelligent defaults for the prime and base numbers needed for the calculation, while allowing you to supply your own. It requires that the openssl binary be installed on the system on which this is run, as it uses that to handle the encryption and decryption. If openssl is not available, a RuntimeError will be raised.

compute_shared(*other*)

decrypt(*text*)

encrypt(*text*)

generate_private()

get_public()

class XenAPIBasedAgent(*session*, *virtapi*, *instance*, *vm_ref*)
Bases: object

get_version()

inject_file(*path*, *contents*)

inject_files(*injected_files*)

inject_ssh_key()

resetnetwork()

set_admin_password(*new_pass*)
Set the root/admin password on the VM instance.

This is done via an agent running on the VM. Communication between nova and the agent is done via writing xenstore records. Since communication is done over the XenAPI RPC calls, we need to encrypt the password. We're using a simple Diffie-Hellman class instead of a more advanced library (such as M2Crypto) for compatibility with the agent code.

update_if_needed(*version*)

find_guest_agent(*base_dir*)
tries to locate a guest agent at the path specified by agent_rel_path

is_upgrade_required(*current_version*, *available_version*)

should_use_agent(*instance*)

3.7.718 The nova.virt.xenapi.client.objects Module

class Host(*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

XenServer hosts.

class Network(*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Networks that VIFs are attached to.

class PBD(*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Physical block device.

class PIF (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Physical Network Interface.

class Pool (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Pool of hosts.

class SR (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Storage Repository.

class VBD (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Virtual block device.

plug (*vbd_ref*, *vm_ref*)

unplug (*vbd_ref*, *vm_ref*)

class VDI (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Virtual disk image.

class VLAN (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

VLAN.

class VM (*session*)
Bases: nova.virt.xenapi.client.objects.XenAPISessionObject

Virtual Machine.

class XenAPISessionObject (*session*, *name*)
Bases: object

Wrapper to make calling and mocking the session easier

The XenAPI protocol is an XML RPC API that is based around the XenAPI database, and operations you can do on each of the objects stored in the database, such as VM, SR, VDI, etc.

For more details see the XenAPI docs: http://docs.vmd.citrix.com/XenServer/6.2.0/1.0/en_gb/api/

Most, objects like VM, SR, VDI, etc, share a common set of methods: * vm_ref = session.VM.create(vm_rec) * vm_ref = session.VM.get_by_uuid(uuid) * session.VM.destroy(vm_ref) * vm_refs = session.VM.get_all()

Each object also has specific messages, or functions, such as: * session.VM.clean_reboot(vm_ref)

Each object has fields, like "VBDs" that can be fetched like this: * vbd_refs = session.VM.get_VBDs(vm_ref)

You can get all the fields by fetching the full record. However please note this is much more expensive than just fetching the field you require: * vm_rec = session.VM.get_record(vm_ref)

When searching for particular objects, you may be tempted to use get_all(), but this often leads to races as objects get deleted under your feet. It is preferable to use the undocumented: * vms = session.VM.get_all_records_where('field "is_control_domain"="true"')

3.7.719 The nova.virt.xenapi.client.session Module

class XenAPISession(*url*, *user*, *pw*)
Bases: object

The session to invoke XenAPI SDK calls.

PLUGIN_REQUIRED_VERSION = '1.2'

call_plugin(*plugin*, *fn*, *args*)
Call host.call_plugin on a background thread.

call_plugin_serialized(*plugin*, *fn*, **args*, ***kwargs*)

call_plugin_serialized_with_retry(*plugin*, *fn*, *num_retries*, *callback*, *retry_cb=None*, **args*, ***kwargs*)
Allows a plugin to raise RetryableError so we can try again.

call_xenapi(*method*, **args*)
Call the specified XenAPI method on a background thread.

get_all_refs_and_recs(*record_type*)
Retrieve all refs and recs for a Xen record type.

Handles race-conditions where the record may be deleted between the *get_all* call and the *get_record* call.

get_rec(*record_type*, *ref*)

get_session_id()
Return a string session_id. Used for vnc consoles.

apply_session_helpers(*session*)

3.7.720 The nova.virt.xenapi.driver Module

A driver for XenServer or Xen Cloud Platform.

Variable Naming Scheme

- suffix "_ref" for opaque references
- suffix "_uuid" for UUIDs
- suffix "_rec" for record objects

class XenAPIDriver(*virtapi*, *read_only=False*)
Bases: nova.virt.driver.ComputeDriver

A connection to XenServer or Xen Cloud Platform.

add_to_aggregate(*context*, *aggregate*, *host*, ***kwargs*)
Add a compute host to an aggregate.

attach_volume(*context*, *connection_info*, *instance*, *mountpoint*, *disk_bus=None*, *device_type=None*, *encryption=None*)
Attach volume storage to VM instance.

change_instance_metadata(*context*, *instance*, *diff*)
Apply a diff to the instance metadata.

check_can_live_migrate_destination(*context*, *instance*, *src_compute_info*, *dst_compute_info*, *block_migration=False*, *disk_over_commit=False*)
Check if it is possible to execute live migration.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance object
- **block_migration** – if true, prepare for block migration
- **disk_over_commit** – if true, allow disk over commit

check_can_live_migrate_destination_cleanup(*context*, *dest_check_data*)
Do required cleanup on dest host after check_can_live_migrate calls

Parameters

- **context** – security context
- **dest_check_data** – result of check_can_live_migrate_destination

check_can_live_migrate_source(*context*, *instance*, *dest_check_data*, *block_device_info=None*)
Check if it is possible to execute live migration.

This checks if the live migration can succeed, based on the results from check_can_live_migrate_destination.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance
- **dest_check_data** – result of check_can_live_migrate_destination includes the block_migration flag
- **block_device_info** – result of _get_instance_block_device_info

cleanup(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*, *destroy_vifs=True*)
Cleanup after instance being destroyed by Hypervisor.

confirm_migration(*migration*, *instance*, *network_info*)
Confirms a resize, destroying the source VM.

destroy(*context*, *instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*, *migrate_data=None*)
Destroy VM instance.

detach_volume(*connection_info*, *instance*, *mountpoint*, *encryption=None*)
Detach volume storage from VM instance.

ensure_filtering_rules_for_instance(*instance*, *network_info*)

estimate_instance_overhead(*instance_info*)
Get virtualization overhead required to build an instance of the given flavor.

Parameters **instance_info** – Instance/flavor to calculate overhead for.

Returns Overhead memory in MB.

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)
Completes a resize, turning on the migrated instance.

finish_revert_migration(*context*, *instance*, *network_info*, *block_device_info=None*, *power_on=True*)
Finish reverting a resize.

get_all_bw_counters(*instances*)
Return bandwidth usage counters for each interface on each running VM.

get_available_nodes(*refresh=False*)

get_available_resource(*nodename*)
Retrieve resource information.

This method is called when nova-compute launches, and as part of a periodic task that records the results in the DB.

Parameters **nodename** – ignored in this driver

Returns dictionary describing resources

get_console_output(*context*, *instance*)
Return snapshot of console.

get_console_pool_info(*console_type*)

get_diagnostics(*instance*)
Return data about VM diagnostics.

get_host_ip_addr()

get_host_uptime()
Returns the result of calling "uptime" on the target host.

get_info(*instance*)
Return data about VM instance.

get_instance_diagnostics(*instance*)
Return data about VM diagnostics.

get_instance_disk_info(*instance*, *block_device_info=None*)
Used by libvirt for live migration. We rely on xenapi checks to do this for us.

get_per_instance_usage()
Get information about instance resource usage.

Returns dict of nova uuid => dict of usage info

get_vnc_console(*context*, *instance*)
Return link to instance's VNC console.

get_volume_connector(*instance*)
Return volume connector information.

host_maintenance_mode(*host*, *mode*)
Start/Stop host maintenance window. On start, it triggers guest VMs evacuation.

host_power_action(*action*)
The only valid values for 'action' on XenServer are 'reboot' or 'shutdown', even though the API also accepts 'startup'. As this is not technically possible on XenServer, since the host is the same physical machine as the hypervisor, if this is requested, we need to raise an exception.

host_state

init_host(*host*)

inject_file(*instance*, *b64_path*, *b64_contents*)
Create a file on the VM instance. The file path and contents should be base64-encoded.

inject_network_info(*instance*, *nw_info*)
inject network info for specified instance.

instance_exists(*instance*)
Checks existence of an instance on the host.

Parameters **instance** – The instance to lookup

Returns True if supplied instance exists on the host, False otherwise.

NOTE(belliott): This is an override of the base method for efficiency.

list_instance_uuids()
Get the list of nova instance uuids for VMs found on the hypervisor.

list_instances()
List VM instances.

live_migration(*context*, *instance*, *dest*, *post_method*, *recover_method*, *block_migration=False*, *migrate_data=None*)
Performs the live migration of the specified instance.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance object instance object that is migrated.
- **dest** – destination host
- **post_method** – post operation method. expected nova.compute.manager._post_live_migration.
- **recover_method** – recovery method when any exception occurs. expected nova.compute.manager._rollback_live_migration.
- **block_migration** – if true, migrate VM disk.
- **migrate_data** – implementation specific params

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *network_info*, *block_device_info=None*, *timeout=0*, *retry_interval=0*)
Transfers the VHD of a running instance to another host, then shuts off the instance copies over the COW disk

pause(*instance*)
Pause VM instance.

plug_vifs(*instance*, *network_info*)
Plug VIFs into networks.

poll_rebooting_instances(*timeout*, *instances*)
Poll for rebooting instances.

post_interrupted_snapshot_cleanup(*context*, *instance*)
Cleans up any resources left after a failed snapshot.

post_live_migration(*context*, *instance*, *block_device_info*, *migrate_data=None*)
Post operation of live migration at source host.

Parameters

- **context** – security context
- **migrate_data** – if not None, it is a dict which has data

Instance instance object that was migrated

Block_device_info instance block device information

post_live_migration_at_destination(*context*, *instance*, *network_info*, *block_migration=False*, *block_device_info=None*)
Post operation of live migration at destination host.

Parameters

- **context** – security context
- **instance** – nova.db.sqlalchemy.models.Instance object instance object that is migrated.
- **network_info** – instance network information
- **block_migration** – if true, post operation of block_migration.

power_off(*instance*, *timeout=0*, *retry_interval=0*)
Power off the specified instance.

power_on(*context*, *instance*, *network_info*, *block_device_info=None*)
Power on the specified instance.

pre_live_migration(*context*, *instance*, *block_device_info*, *network_info*, *disk_info*, *migrate_data=None*)
Preparation live migration.

Parameters **block_device_info** – It must be the result of _get_instance_volume_bdms() at compute manager.

reboot(*context*, *instance*, *network_info*, *reboot_type*, *block_device_info=None*, *bad_volumes_callback=None*)
Reboot VM instance.

refresh_instance_security_rules(*instance*)
Updates security group rules for specified instance.

Invoked when instances are added/removed to a security group or when a rule is added/removed to a security group.

refresh_provider_fw_rules()

refresh_security_group_members(*security_group_id*)
Updates security group rules for all instances associated with a given security group.

Invoked when instances are added/removed to a security group.

refresh_security_group_rules(*security_group_id*)
Updates security group rules for all instances associated with a given security group.

Invoked when security group rules are updated.

remove_from_aggregate(*context*, *aggregate*, *host*, ***kwargs*)
Remove a compute host from an aggregate.

rescue(*context*, *instance*, *network_info*, *image_meta*, *rescue_password*)
Rescue the specified instance.

reset_network(*instance*)
reset networking for specified instance.

restore(*instance*)
Restore the specified instance.

resume(*context*, *instance*, *network_info*, *block_device_info=None*)
resume the specified instance.

resume_state_on_host_boot(*context*, *instance*, *network_info*, *block_device_info=None*)
resume guest state when a host is booted.

rollback_live_migration_at_destination(*context*, *instance*, *network_info*, *block_device_info*, *destroy_disks=True*, *migrate_data=None*)

set_admin_password(*instance*, *new_pass*)
Set the root/admin password on the VM instance.

set_bootable(*instance*, *is_bootable*)
Set the ability to power on/off an instance.

set_host_enabled(*enabled*)
Sets the compute host's ability to accept new instances.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Create snapshot from a running VM instance.

soft_delete(*instance*)
Soft delete the specified instance.

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*)
Create VM instance.

suspend(*context*, *instance*)
suspend the specified instance.

undo_aggregate_operation(*context*, *op*, *aggregate*, *host*, *set_error=True*)
Undo aggregate operation when pool error raised.

unfilter_instance(*instance*, *network_info*)
Removes security groups configured for an instance.

unpause(*instance*)
Unpause paused VM instance.

unplug_vifs(*instance*, *network_info*)
Unplug VIFs from networks.

unrescue(*instance*, *network_info*)
Unrescue the specified instance.

3.7.721 The nova.virt.xenapi.fake Module

A fake XenAPI SDK.

exception Failure(*details*)
Bases: exceptions.Exception

class FakeXenAPI
Bases: object

class SessionBase(*uri*)
Bases: object

Base class for Fake Sessions.

PBD_create(*_1*, *pbd_rec*)

PBD_plug(*_1*, *pbd_ref*)

PBD_unplug(*_1*, *pbd_ref*)

SR_forget(*_1*, *sr_ref*)

SR_introduce(*_1*, *sr_uuid*, *label*, *desc*, *type*, *content_type*, *shared*, *sm_config*)

SR_scan(*_1*, *sr_ref*)

VBD_add_to_other_config(*_1*, *vbd_ref*, *key*, *value*)

VBD_get_other_config(*_1*, *vbd_ref*)

VBD_insert(*_1*, *vbd_ref*, *vdi_ref*)

VBD_plug(*_1*, *ref*)

VBD_unplug(*_1*, *ref*)

VDI_add_to_other_config(*_1*, *vdi_ref*, *key*, *value*)

VDI_clone(*_1*, *vdi_to_clone_ref*)

VDI_copy(*_1*, *vdi_to_copy_ref*, *sr_ref*)

VDI_get_virtual_size(**args*)

VDI_remove_from_other_config(*_1*, *vdi_ref*, *key*)

VDI_resize(**args*)

VDI_resize_online(**args*)

VM_add_to_xenstore_data(*_1*, *vm_ref*, *key*, *value*)

VM_assert_can_migrate(*session*, *vmref*, *migrate_data*, *live*, *vdi_map*, *vif_map*, *options*)

VM_clean_reboot(*session*, *vm_ref*)

VM_clean_shutdown(*session*, *vm_ref*)

VM_get_xenstore_data(*_1*, *vm_ref*)

VM_hard_reboot(*session*, *vm_ref*)

VM_hard_shutdown(*session*, *vm_ref*)

VM_migrate_send(*session*, *mref*, *migrate_data*, *live*, *vdi_map*, *vif_map*, *options*)

VM_pause(*session*, *vm_ref*)

VM_pool_migrate(*_1*, *vm_ref*, *host_ref*, *options*)

VM_remove_from_blocked_operations(*session*, *vm_ref*, *key*)

VM_remove_from_xenstore_data(*_1*, *vm_ref*, *key*)

VM_suspend(*session*, *vm_ref*)

host_call_plugin(*_1*, *_2*, *plugin*, *method*, *args*)

host_compute_free_memory(*_1*, *ref*)

host_migrate_receive(*session*, *destref*, *nwref*, *options*)

pool_eject(*session*, *host_ref*)

pool_get_default_SR(*_1*, *pool_ref*)

pool_join(*session*, *hostname*, *username*, *password*)

pool_set_name_label(*session*, *pool_ref*, *name*)

xenapi_request(*methodname*, *params*)

after_VBD_create(*vbd_ref*, *vbd_rec*)
Create read-only fields and backref from VM and VDI to VBD when VBD is created.

after_VDI_create(*vdi_ref*, *vdi_rec*)

after_VIF_create(*vif_ref*, *vif_rec*)
Create backref from VM to VIF when VIF is created.

after_VM_create(*vm_ref*, *vm_rec*)
Create read-only fields in the VM record.

as_json(**args*, ***kwargs*)
Helper function for simulating XenAPI plugin responses for those that are returning JSON. If this function is given plain arguments, then these are rendered as a JSON list. If it's given keyword arguments then these are rendered as a JSON dict.

as_value(*s*)
Helper function for simulating XenAPI plugin responses. It escapes and wraps the given argument.

check_for_session_leaks()

create_host(*name_label*, *hostname='fake_name'*, *address='fake_addr'*)

create_network(*name_label*, *bridge*)

create_pbd(*host_ref*, *sr_ref*, *attached*)

create_sr(***kwargs*)

create_task(*name_label*)

create_vbd(*vm_ref*, *vdi_ref*, *userdevice=0*, *other_config=None*)

create_vdi(*name_label*, *sr_ref*, ***kwargs*)

create_vm(*name_label*, *status*, ***kwargs*)

destroy_vbd(*vbd_ref*)

destroy_vdi(*vdi_ref*)

destroy_vm(*vm_ref*)

get_all(*table*)

get_all_records(*table*)

get_all_records_where(*table_name*, *query*)

get_record(*table*, *ref*)

reset()

reset_table(*table*)

3.7.722 The nova.virt.xenapi.firewall Module

class Dom0IptablesFirewallDriver(*virtapi*, *xenapi_session=None*, ***kwargs*)
Bases: nova.virt.firewall.IptablesFirewallDriver

Dom0IptablesFirewallDriver class

This class provides an implementation for nova.virt.Firewall using iptables. This class is meant to be used with the xenapi backend and uses xenapi plugin to enforce iptables rules in dom0.

3.7.723 The nova.virt.xenapi.host Module

Management class for host-related functions (start, reboot, etc).

class Host(*session*, *virtapi*)
Bases: object

Implements host related operations.

get_host_uptime()
Returns the result of calling "uptime" on the target host.

host_maintenance_mode(*host*, *mode*)
Start/Stop host maintenance window. On start, it triggers guest VMs evacuation.

host_power_action(*action*)
Reboots or shuts down the host.

set_host_enabled(*enabled*)
Sets the compute host's ability to accept new instances.

class HostState(*session*)
Bases: object

Manages information about the XenServer host this compute node is running on.

get_host_stats(*refresh=False*)
Return the current state of the host. If 'refresh' is True, run the update first.

update_status()
Since under Xenserver, a compute node runs on a given host, we can get host status information using xenapi.

call_xenhost(*session*, *method*, *arg_dict*)
There will be several methods that will need this general handling for interacting with the xenhost plugin, so this abstracts out that behavior.

to_cpu_model(*host_cpu_info*)

to_supported_instances(*host_capabilities*)

3.7.724 The nova.virt.xenapi.image.bittorrent Module

class BittorrentStore
Bases: object

download_image(*context*, *session*, *instance*, *image_id*)

upload_image(*context*, *session*, *instance*, *image_id*, *vdi_uuids*)

3.7.725 The nova.virt.xenapi.image.glance Module

class GlanceStore
Bases: object

download_image(*context*, *session*, *instance*, *image_id*)

upload_image(*context*, *session*, *instance*, *image_id*, *vdi_uuids*)

3.7.726 The nova.virt.xenapi.image.utils Module

class GlanceImage(*context*, *image_href_or_id*)
Bases: object

data()

download_to(*fileobj*)

is_raw_tgz()

meta

class IterableToFileAdapter(*iterable*)
Bases: object

A degenerate file-like so that an iterable could be read like a file.

As Glance client returns an iterable, but tarfile requires a file like, this is the adapter between the two. This allows tarfile to access the glance stream.

read(*size*)

class RawImage(*glance_image*)
Bases: object

get_size()

stream_to(*fileobj*)

class RawTGZImage(*glance_image*)
Bases: object

get_size()

stream_to(*target_file*)

3.7.727 The nova.virt.xenapi.image.vdi_through_dev Module

class TarGzProducer(*devpath*, *writefile*, *size*, *fname*)
Bases: object

get_metadata()

start()

class UploadToGlanceAsRawTgz(*context*, *session*, *instance*, *image_id*, *vdi_uuids*)
Bases: object

upload_image()

class VdiThroughDevStore
Bases: object

Deal with virtual disks by attaching them to the OS domU.

At the moment it supports upload to Glance, and the upload format is a raw disk inside a tgz.

download_image(*context*, *session*, *instance*, *image_id*)

upload_image(*context*, *session*, *instance*, *image_id*, *vdi_uuids*)

3.7.728 The nova.virt.xenapi.network_utils Module

Helper methods for operations related to the management of network records and their attributes like bridges, PIFs, QoS, as well as their lookup functions.

find_network_with_bridge(*session*, *bridge*)
Return the network on which the bridge is attached, if found. The bridge is defined in the nova db and can be found either in the 'bridge' or 'name_label' fields of the XenAPI network record.

find_network_with_name_label(*session*, *name_label*)

3.7.729 The nova.virt.xenapi.pool Module

Management class for Pool-related functions (join, eject, etc).

class ResourcePool(*session*, *virtapi*)
Bases: object

Implements resource pool operations.

add_to_aggregate(*context*, *aggregate*, *host*, *slave_info=None*)
Add a compute host to an aggregate.

remove_from_aggregate(*context*, *aggregate*, *host*, *slave_info=None*)
Remove a compute host from an aggregate.

undo_aggregate_operation(*context*, *op*, *aggregate*, *host*, *set_error*)
Undo aggregate operation when pool error raised.

swap_xapi_host(*url*, *host_addr*)
Replace the XenServer address present in 'url' with 'host_addr'.

3.7.730 The nova.virt.xenapi.pool_states Module

Possible states for xen resource pools.

A pool may be 'created', in which case the admin has triggered its creation, but the underlying hypervisor pool has not actually being set up yet. A pool may be 'changing', meaning that the underlying hypervisor pool is being setup. A pool may be 'active', in which case the underlying hypervisor pool is up and running. A pool may be 'dismissed' when it has no hosts and it has been deleted. A pool may be in 'error' in all other cases. A 'created' pool becomes 'changing' during the first request of adding a host. During a 'changing' status no other requests will be accepted; this is to allow the hypervisor layer to instantiate the underlying pool without any potential race condition that may incur in master/slave-based configurations. The pool goes into the 'active' state when the underlying pool has been correctly instantiated. All other operations (e.g. add/remove hosts) that succeed will keep the pool in the 'active' state. If a number of continuous requests fail, an 'active' pool goes into an 'error' state. To recover from such a state, admin intervention is required. Currently an error state is irreversible, that is, in order to recover from it a pool must be deleted.

is_hv_pool(*metadata*)
Checks if aggregate is a hypervisor_pool.

3.7.731 The nova.virt.xenapi.vif Module

VIF drivers for XenAPI.

class **XenAPIBridgeDriver** (*xenapi_session*)
Bases: nova.virt.xenapi.vif.XenVIFDriver

VIF Driver for XenAPI that uses XenAPI to create Networks.

plug (*instance*, *vif*, *vm_ref=None*, *device=None*)

unplug (*instance*, *vif*)

class **XenAPIOpenVswitchDriver** (*xenapi_session*)
Bases: nova.virt.xenapi.vif.XenVIFDriver

VIF driver for Open vSwitch with XenAPI.

plug (*instance*, *vif*, *vm_ref=None*, *device=None*)

unplug (*instance*, *vif*)

class **XenVIFDriver** (*xenapi_session*)
Bases: object

3.7.732 The nova.virt.xenapi.vm_utils Module

Helper methods for operations related to the management of VM records and their attributes like VDIs, VIFs, as well as their lookup functions.

class **ImageType**
Bases: object

Enumeration class for distinguishing different image types

0 - kernel image (goes on dom0's filesystem)
1 - ramdisk image (goes on dom0's filesystem)
2 - disk image (local SR, partitioned by objectstore plugin)
3 - raw disk image (local SR, NOT partitioned by plugin)
4 - vhd disk image (local SR, NOT inspected by XS, PV assumed for
linux, HVM assumed for Windows)
5 - ISO disk image (local SR, NOT partitioned by plugin)
6 - config drive

DISK = 2

DISK_CONFIGDRIVE = 6

DISK_CONFIGDRIVE_STR = 'configdrive'

DISK_ISO = 5

DISK_ISO_STR = 'iso'

DISK_RAW = 3

DISK_RAW_STR = 'os_raw'

DISK_STR = 'root'

DISK_VHD = 4

DISK_VHD_STR = 'vhd'

KERNEL = 0

KERNEL_STR = 'kernel'

RAMDISK = 1

RAMDISK_STR = 'ramdisk'

classmethod get_role(*image_type_id*)
Get the role played by the image, based on its type.

classmethod to_string(*image_type*)

attach_cd(*session*, *vm_ref*, *vdi_ref*, *userdevice*)
Create an empty VBD, then insert the CD.

clean_shutdown_vm(*session*, *instance*, *vm_ref*)

cleanup_attached_vdis(*session*)
Unplug any instance VDIs left after an unclean restart.

compile_diagnostics(*vm_rec*)
Compile VM diagnostics data.

compile_info(*session*, *vm_ref*)
Fill record with VM status information.

compile_instance_diagnostics(*instance*, *vm_rec*)

create_image(*context*, *session*, *instance*, *name_label*, *image_id*, *image_type*)
Creates VDI from the image stored in the local cache. If the image is not present in the cache, it streams it from glance.

Returns: A list of dictionaries that describe VDIs

create_kernel_and_ramdisk(*context*, *session*, *instance*, *name_label*)

create_vbd(*session*, *vm_ref*, *vdi_ref*, *userdevice*, *vbd_type='disk'*, *read_only=False*, *bootable=False*, *osvol=False*, *empty=False*, *unpluggable=True*)
Create a VBD record and returns its reference.

create_vdi(*session*, *sr_ref*, *instance*, *name_label*, *disk_type*, *virtual_size*, *read_only=False*)
Create a VDI record and returns its reference.

create_vm(*session*, *instance*, *name_label*, *kernel*, *ramdisk*, *use_pv_kernel=False*, *device_id=None*)
Create a VM record. Returns new VM reference. the use_pv_kernel flag indicates whether the guest is HVM or PV

There are 3 scenarios:

1.Using paravirtualization, kernel passed in

2.Using paravirtualization, kernel within the image

3.Using hardware virtualization

destroy_cached_images(*session*, *sr_ref*, *all_cached=False*, *dry_run=False*)
Destroy used or unused cached images.

A cached image that is being used by at least one VM is said to be 'used'.

In the case of an 'unused' image, the cached image will be the only descendent of the base-copy. So when we delete the cached-image, the refcount will drop to zero and XenServer will automatically destroy the base-copy for us.

The default behavior of this function is to destroy only 'unused' cached images. To destroy all cached images, use the *all_cached=True* kwarg.

destroy_kernel_ramdisk(*session*, *instance*, *kernel*, *ramdisk*)

destroy_vbd(*session*, *vbd_ref*)
Destroy VBD from host database.

destroy_vdi(*session*, *vdi_ref*)

destroy_vm(*session*, *instance*, *vm_ref*)
Destroys a VM record.

determine_disk_image_type(*image_meta*)
Disk Image Types are used to determine where the kernel will reside within an image. To figure out which type we're dealing with, we use the following rules:

1.If we're using Glance, we can use the image_type field to determine the image_type

2.If we're not using Glance, then we need to deduce this based on whether a kernel_id is specified.

determine_vm_mode(*instance*, *disk_image_type*)

ensure_correct_host(*session*)
Ensure we're connected to the host we're running on. This is the required configuration for anything that uses vdi_attached_here.

fetch_bandwidth(*session*)

generate_configdrive(*session*, *instance*, *vm_ref*, *userdevice*, *network_info*, *admin_password=None*, *files=None*)

generate_ephemeral(*session*, *instance*, *vm_ref*, *first_userdevice*, *instance_name_label*, *total_size_gb*)

generate_iso_blank_root_disk(*session*, *instance*, *vm_ref*, *userdevice*, *name_label*, *size_gb*)

generate_single_ephemeral(*session*, *instance*, *vm_ref*, *userdevice*, *size_gb*, *instance_name_label=None*)

generate_swap(*session*, *instance*, *vm_ref*, *userdevice*, *name_label*, *swap_mb*)

get_all_vdi_uuids_for_vm(*session*, *vm_ref*, *min_userdevice=0*)

get_compression_level()

get_ephemeral_disk_sizes(*total_size_gb*)

get_instance_vdis_for_sr(*session*, *vm_ref*, *sr_ref*)
Return opaqueRef for all the vdis which live on sr.

get_power_state(*session*, *vm_ref*)

get_sr_path(*session*, *sr_ref=None*)
Return the path to our storage repository

This is used when we're dealing with VHDs directly, either by taking snapshots or by restoring an image in the DISK_VHD format.

get_this_vm_uuid(*session*)

get_vdi_for_vm_safely(*session*, *vm_ref*, *userdevice='0'*)
Retrieves the primary VDI for a VM.

get_vm_device_id(*session*, *image_properties*)

handle_ipxe_iso(*session*, *instance*, *cd_vdi*, *network_info*)
iPXE ISOs are a mechanism to allow the customer to roll their own image.

To use this feature, a service provider needs to configure the appropriate Nova flags, roll an iPXE ISO, then distribute that image to customers via Glance.

NOTE: *mkisofs* is not present by default in the Dom0, so the service provider can either add that package manually to Dom0 or include the *mkisofs* binary in the image itself.

hard_shutdown_vm(*session*, *instance*, *vm_ref*)

import_all_migrated_disks(*session*, *instance*, *import_root=True*)

is_enough_free_mem(*session*, *instance*)

is_snapshot(*session*, *vm*)

is_vm_shutdown(*session*, *vm_ref*)

list_vms(*session*)

lookup(*session*, *name_label*, *check_rescue=False*)
Look the instance up and return it if available. :param:check_rescue: if True will return the 'name'-rescue vm if it exists, instead of just 'name'

lookup_kernel_ramdisk(*session*, *vm*)

lookup_vm_vdis(*session*, *vm_ref*)
Look for the VDIs that are attached to the VM.

migrate_vhd(*session*, *instance*, *vdi_uuid*, *dest*, *sr_path*, *seq_num*, *ephemeral_number=0*)

preconfigure_instance(*session*, *instance*, *vdi_ref*, *network_info*)
Makes alterations to the image before launching as part of spawn.

remove_old_snapshots(*session*, *instance*, *vm_ref*)
See if there is an snapshot present that should be removed.

resize_disk(*session*, *instance*, *vdi_ref*, *flavor*)

safe_destroy_vdis(*session*, *vdi_refs*)
Tries to destroy the requested VDIs, but ignores any errors.

safe_find_sr(*session*)
Same as _find_sr except raises a NotFound exception if SR cannot be determined

scan_default_sr(*session*)
Looks for the system default SR and triggers a re-scan.

set_other_config_pci(*session*, *vm_ref*, *params*)
Set the pci key of other-config parameter to params.

set_vm_name_label(*session*, *vm_ref*, *name_label*)

snapshot_attached_here(**args*, ***kwds*)

strip_base_mirror_from_vdis(*session*, *vm_ref*)

try_auto_configure_disk(*session*, *vdi_ref*, *new_gb*)

unplug_vbd(*session*, *vbd_ref*, *this_vm_ref*)

update_vdi_virtual_size(*session*, *instance*, *vdi_ref*, *new_gb*)

vdi_attached_here(**args*, ***kwds*)

vm_ref_or_raise(*session*, *instance_name*)

3.7.733 The nova.virt.xenapi.vmops Module

Management class for VM-related functions (spawn, reboot, etc).

class VMOps(*session*, *virtapi*)
Bases: object

Management class for VM-related tasks.

agent_enabled(*instance*)

change_instance_metadata(*instance*, *diff*)
Apply changes to instance metadata to xenstore.

check_can_live_migrate_destination(*ctxt*, *instance_ref*, *block_migration=False*, *disk_over_commit=False*)
Check if it is possible to execute live migration.

Parameters

- **ctxt** – security context
- **instance_ref** – nova.db.sqlalchemy.models.Instance object
- **block_migration** – if true, prepare for block migration
- **disk_over_commit** – if true, allow disk over commit

check_can_live_migrate_source(*ctxt*, *instance_ref*, *dest_check_data*)
Check if it's possible to execute live migration on the source side.

Parameters

- **ctxt** – security context
- **instance_ref** – nova.db.sqlalchemy.models.Instance object
- **dest_check_data** – data returned by the check on the destination, includes block_migration flag

confirm_migration(*migration*, *instance*, *network_info*)

connect_block_device_volumes(*block_device_info*)

destroy(*instance*, *network_info*, *block_device_info=None*, *destroy_disks=True*)
Destroy VM instance.

This is the method exposed by xenapi_conn.destroy(). The rest of the destroy_* methods are internal.

finish_migration(*context*, *migration*, *instance*, *disk_info*, *network_info*, *image_meta*, *resize_instance*, *block_device_info=None*, *power_on=True*)

finish_revert_migration(*context*, *instance*, *block_device_info=None*, *power_on=True*)

get_all_bw_counters()
Return running bandwidth counter for each interface on each running VM.

get_console_output(*instance*)
Return last few lines of instance console.

get_diagnostics(*instance*)
Return data about VM diagnostics.

get_info(*instance*, *vm_ref=None*)
Return data about VM instance.

get_instance_diagnostics(*instance*)
Return data about VM diagnostics using the common API.

get_per_instance_usage()
Get usage info about each active instance.

get_vnc_console(*instance*)
Return connection info for a vnc console.

inject_file(*instance*, *path*, *contents*)
Write a file to the VM instance.

inject_network_info(*instance*, *network_info*, *vm_ref=None*)
Generate the network info and make calls to place it into the xenstore and the xenstore param list. vm_ref can be passed in because it will sometimes be different than what vm_utils.lookup(session, instance['name']) will find (ex: rescue)

instance_exists(*name_label*)

list_instance_uuids()
Get the list of nova instance uuids for VMs found on the hypervisor.

list_instances()
List VM instances.

live_migrate(*context*, *instance*, *destination_hostname*, *post_method*, *recover_method*, *block_migration*, *migrate_data=None*)

migrate_disk_and_power_off(*context*, *instance*, *dest*, *flavor*, *block_device_info*)
Copies a VHD from one host machine to another, possibly resizing filesystem before hand.

Parameters

- **instance** – the instance that owns the VHD in question.
- **dest** – the destination host machine.
- **flavor** – flavor to resize to

pause(*instance*)
Pause VM instance.

plug_vifs(*instance*, *network_info*)
Set up VIF networking on the host.

poll_rebooting_instances(*timeout*, *instances*)
Look for expirable rebooting instances.

•issue a "hard" reboot to any instance that has been stuck in a reboot state for >= the given timeout

post_interrupted_snapshot_cleanup(*context*, *instance*)
Cleans up any resources left after a failed snapshot.

post_live_migration(*context*, *instance*, *migrate_data=None*)

post_live_migration_at_destination(*context*, *instance*, *network_info*, *block_migration*, *block_device_info*)

power_off(*instance*)
Power off the specified instance.

power_on(*instance*)
Power on the specified instance.

reboot(*instance*, *reboot_type*, *bad_volumes_callback=None*)
Reboot VM instance.

refresh_instance_security_rules(*instance*)
recreates security group rules for specified instance.

refresh_provider_fw_rules()

refresh_security_group_members(*security_group_id*)
recreates security group rules for every instance.

refresh_security_group_rules(*security_group_id*)
recreates security group rules for every instance.

rescue(*context*, *instance*, *network_info*, *image_meta*, *rescue_password*)
Rescue the specified instance.

•shutdown the instance VM.

•set 'bootlock' to prevent the instance from starting in rescue.

•spawn a rescue VM (the vm name-label will be instance-N-rescue).

reset_network(*instance*, *rescue=False*)
Calls resetnetwork method in agent.

restore(*instance*)
Restore the specified instance.

resume(*instance*)
Resume the specified instance.

set_admin_password(*instance*, *new_pass*)
Set the root/admin password on the VM instance.

set_bootable(*instance*, *is_bootable*)
Set the ability to power on/off an instance.

snapshot(*context*, *instance*, *image_id*, *update_task_state*)
Create snapshot from a running VM instance.

Parameters

- **context** – request context
- **instance** – instance to be snapshotted
- **image_id** – id of image to upload to

Steps involved in a XenServer snapshot:

1. XAPI-Snapshot: Snapshotting the instance using XenAPI. This creates: Snapshot (Template) VM, Snapshot VBD, Snapshot VDI, Snapshot VHD
2. Wait-for-coalesce: The Snapshot VDI and Instance VDI both point to a 'base-copy' VDI. The base_copy is immutable and may be chained with other base_copies. If chained, the base_copies coalesce together, so, we must wait for this coalescing to occur to get a stable representation of the data on disk.
3. Push-to-data-store: Once coalesced, we call 'image_upload_handler' to upload the images.

soft_delete(*instance*)
Soft delete the specified instance.

spawn(*context*, *instance*, *image_meta*, *injected_files*, *admin_password*, *network_info=None*, *block_device_info=None*, *name_label=None*, *rescue=False*)

suspend(*instance*)
Suspend the specified instance.

unfilter_instance(*instance_ref*, *network_info*)
Removes filters for each VIF of the specified instance.

unpause(*instance*)
Unpause VM instance.

unplug_vifs(*instance*, *network_info*)

unrescue(*instance*)
Unrescue the specified instance.

- unplug the instance VM's disk from the rescue VM.
- teardown the rescue VM.
- release the bootlock to allow the instance VM to start.

make_step_decorator(*context*, *instance*, *update_instance_progress*, *total_offset=0*)
Factory to create a decorator that records instance progress as a series of discrete steps.

Each time the decorator is invoked we bump the total-step-count, so after:

```
@step
def step1():
    ...

@step
def step2():
    ...
```

we have a total-step-count of 2.

Each time the step-function (not the step-decorator!) is invoked, we bump the current-step-count by 1, so after:

```
step1()
```

the current-step-count would be 1 giving a progress of `1 / 2 * 100` or 50%.

3.7.734 The nova.virt.xenapi.volume_utils Module

Helper methods for operations related to the management of volumes, and storage repositories

find_sr_by_uuid(*session*, *sr_uuid*)
Return the storage repository given a uuid.

find_sr_from_vbd(*session*, *vbd_ref*)
Find the SR reference from the VBD reference.

find_sr_from_vdi(*session*, *vdi_ref*)
Find the SR reference from the VDI reference.

find_vbd_by_number(*session*, *vm_ref*, *dev_number*)
Get the VBD reference from the device number.

forget_sr(*session*, *sr_ref*)
Forgets the storage repository without destroying the VDIs within.

get_device_number(*mountpoint*)

introduce_sr(*session*, *sr_uuid*, *label*, *params*)

introduce_vdi(*session*, *sr_ref*, *vdi_uuid=None*, *target_lun=None*)
Introduce VDI in the host.

is_booted_from_volume(*session*, *vm_ref*)
Determine if the root device is a volume.

parse_sr_info(*connection_data*, *description=''*)

purge_sr(*session*, *sr_ref*)

3.7.735 The nova.virt.xenapi.volumeops Module

Management class for Storage-related functions (attach, detach, etc).

class VolumeOps(*session*)
Bases: object

Management class for Volume-related tasks.

attach_volume(*connection_info*, *instance_name*, *mountpoint*, *hotplug=True*)
Attach volume to VM instance.

connect_volume(*connection_info*)
Attach volume to hypervisor, but not the VM.

detach_all(*vm_ref*)
Detach all cinder volumes.

detach_volume(*connection_info*, *instance_name*, *mountpoint*)
Detach volume storage to VM instance.

find_bad_volumes(*vm_ref*)
Find any volumes with their connection severed.

Certain VM operations (e.g. *VM.start*, *VM.reboot*, etc.) will not work when a VBD is present that points to a non-working volume. To work around this, we scan for non-working volumes and detach them before retrying a failed operation.

safe_cleanup_from_vdis(*vdi_refs*)

3.7.736 The nova.vnc.xvp_proxy Module

Eventlet WSGI Services to proxy VNC for XCP protocol.

class SafeHttpProtocol(*request*, *client_address*, *server*)
Bases: eventlet.wsgi.HttpProtocol

HttpProtocol wrapper to suppress IOErrors.

The proxy code above always shuts down client connections, so we catch the IOError that raises when the SocketServer tries to flush the connection.

finish()

class XCPVNCProxy
Bases: object

Class to use the xvp auth protocol to proxy instance vnc consoles.

handshake(*req*, *connect_info*, *sockets*)
Execute hypervisor-specific vnc auth handshaking (if needed).

one_way_proxy(*source*, *dest*)
Proxy tcp connection from source to dest.

proxy_connection(*req*, *connect_info*, *start_response*)
Spawn bi-directional vnc proxy.

get_wsgi_server()

3.7.737 The nova.volume.cinder Module

Handles all requests relating to volumes + cinder.

class API

Bases: object

API for interacting with the volume manager.

attach(*ctx*, *volume_id*, **args*, ***kwargs*)

begin_detaching(*ctx*, *volume_id*, **args*, ***kwargs*)

check_attach(*context*, *volume*, *instance=None*)

check_attached(*context*, *volume*)

check_detach(*context*, *volume*)

create(*context*, *size*, *name*, *description*, *snapshot=None*, *image_id=None*, *volume_type=None*, *metadata=None*, *availability_zone=None*)

create_snapshot(*ctx*, *volume_id*, **args*, ***kwargs*)

create_snapshot_force(*ctx*, *volume_id*, **args*, ***kwargs*)

delete(*ctx*, *volume_id*, **args*, ***kwargs*)

delete_snapshot(*ctx*, *snapshot_id*, **args*, ***kwargs*)

detach(*ctx*, *volume_id*, **args*, ***kwargs*)

get(*ctx*, *volume_id*, **args*, ***kwargs*)

get_all(*context*, *search_opts=None*)

get_all_snapshots(*context*)

get_snapshot(*ctx*, *snapshot_id*, **args*, ***kwargs*)

get_volume_encryption_metadata(*context*, *volume_id*)

initialize_connection(*ctx*, *volume_id*, **args*, ***kwargs*)

migrate_volume_completion(*context*, *old_volume_id*, *new_volume_id*, *error=False*)

reserve_volume(*ctx*, *volume_id*, **args*, ***kwargs*)

roll_detaching(*ctx*, *volume_id*, **args*, ***kwargs*)

terminate_connection(*ctx*, *volume_id*, **args*, ***kwargs*)

unreserve_volume(*ctx*, *volume_id*, **args*, ***kwargs*)

update(*ctx*, *volume_id*, **args*, ***kwargs*)

update_snapshot_status(*ctx*, *snapshot_id*, **args*, ***kwargs*)

cinderclient(*context*)

get_cinder_client_version(*url*)

Parse cinder client version by endpoint url.

Parameters **url** – URL for cinder.

Returns str value(1 or 2).

reset_globals()
Testing method to reset globals.

translate_snapshot_exception(*method*)
Transforms the exception for the snapshot but keeps its traceback intact.

translate_volume_exception(*method*)
Transforms the exception for the volume but keeps its traceback intact.

3.7.738 The nova.volume.encryptors.base Module

class VolumeEncryptor(*connection_info*, ***kwargs*)
Bases: object

Base class to support encrypted volumes.

A VolumeEncryptor provides hooks for attaching and detaching volumes, which are called immediately prior to attaching the volume to an instance and immediately following detaching the volume from an instance. This class performs no actions for either hook.

attach_volume(*context*, ***kwargs*)
Hook called immediately prior to attaching a volume to an instance.

detach_volume(***kwargs*)
Hook called immediately after detaching a volume from an instance.

3.7.739 The nova.volume.encryptors.cryptsetup Module

class CryptsetupEncryptor(*connection_info*, ***kwargs*)
Bases: nova.volume.encryptors.base.VolumeEncryptor

A VolumeEncryptor based on dm-crypt.

This VolumeEncryptor uses dm-crypt to encrypt the specified volume.

attach_volume(*context*, ***kwargs*)
Shadows the device and passes an unencrypted version to the instance.

Transparent disk encryption is achieved by mounting the volume via dm-crypt and passing the resulting device to the instance. The instance is unaware of the underlying encryption due to modifying the original symbolic link to refer to the device mounted by dm-crypt.

detach_volume(***kwargs*)
Removes the dm-crypt mapping for the device.

3.7.740 The nova.volume.encryptors.luks Module

class LuksEncryptor(*connection_info*, ***kwargs*)
Bases: nova.volume.encryptors.cryptsetup.CryptsetupEncryptor

A VolumeEncryptor based on LUKS.

This VolumeEncryptor uses dm-crypt to encrypt the specified volume.

attach_volume(*context*, ***kwargs*)
Shadows the device and passes an unencrypted version to the instance.

Transparent disk encryption is achieved by mounting the volume via dm-crypt and passing the resulting device to the instance. The instance is unaware of the underlying encryption due to modifying the original symbolic link to refer to the device mounted by dm-crypt.

is_luks(*device*)
Checks if the specified device uses LUKS for encryption.

Parameters **device** – the device to check

Returns true if the specified device uses LUKS; false otherwise

3.7.741 The nova.volume.encryptors.nop Module

class NoOpEncryptor(*connection_info*, ***kwargs*)
Bases: nova.volume.encryptors.base.VolumeEncryptor

A VolumeEncryptor that does nothing.

This class exists solely to wrap regular (i.e., unencrypted) volumes so that they do not require special handling with respect to an encrypted volume. This implementation performs no action when a volume is attached or detached.

attach_volume(*context*)

detach_volume()

3.7.742 The nova.weights Module

Pluggable Weighing support

class BaseWeigher
Bases: object

Base class for pluggable weighers.

The attributes maxval and minval can be specified to set up the maximum and minimum values for the weighed objects. These values will then be taken into account in the normalization step, instead of taking the values from the calculated weights.

maxval = None

minval = None

weigh_objects(*weighed_obj_list*, *weight_properties*)
Weigh multiple objects.

Override in a subclass if you need access to all objects in order to calculate weights. Do not modify the weight of an object here, just return a list of weights.

weight_multiplier()
How weighted this weigher should be.

Override this method in a subclass, so that the returned value is read from a configuration option to permit operators specify a multiplier for the weigher.

class BaseWeightHandler(*loadable_cls_type*)
Bases: nova.loadables.BaseLoader

get_weighed_objects(*weighers*, *obj_list*, *weighing_properties*)
Return a sorted (descending), normalized list of WeighedObjects.

object_class
alias of WeighedObject

class WeighedObject(*obj*, *weight*)
Bases: object

Object with weight information.

normalize(*weight_list*, *minval=None*, *maxval=None*)
Normalize the values in a list between 0 and 1.0.

The normalization is made regarding the lower and upper values present in weight_list. If the minval and/or maxval parameters are set, these values will be used instead of the minimum and maximum from the list.

If all the values are equal, they are normalized to 0.

3.7.743 The nova.wsgi Module

Utility methods for working with WSGI servers.

class Application
Bases: object

Base WSGI application wrapper. Subclasses need to implement __call__.

classmethod factory(*global_config*, ***local_config*)
Used for paste app factories in paste.deploy config files.

Any local configuration (that is, values under the [app:APPNAME] section of the paste config) will be passed into the *__init__* method as kwargs.

A hypothetical configuration would look like:

[app:wadl] latest_version = 1.3 paste.app_factory = nova.api.fancy_api:Wadl.factory

which would result in a call to the *Wadl* class as

import nova.api.fancy_api fancy_api.Wadl(latest_version='1.3')

You could of course re-implement the *factory* method in subclasses, but using the kwarg passing it shouldn't be necessary.

class Debug(*application*)
Bases: nova.wsgi.Middleware

Helper class for debugging a WSGI application.

Can be inserted into any WSGI application chain to get information about the request and response.

static print_generator(*app_iter*)
Iterator that prints the contents of a wrapper string.

class Loader(*config_path=None*)
Bases: object

Used to load WSGI applications from paste configurations.

load_app(*name*)
Return the paste URLMap wrapped WSGI application.

Parameters **name** – Name of the application to load.

Returns Paste URLMap object wrapping the requested application.

Raises *nova.exception.PasteAppNotFound*

class **Middleware**(*application*)
Bases: nova.wsgi.Application

Base WSGI middleware.

These classes require an application to be initialized that will be called next. By default the middleware will simply call its wrapped app, or you can override __call__ to customize its behavior.

classmethod **factory**(*global_config*, ***local_config*)
Used for paste app factories in paste.deploy config files.

Any local configuration (that is, values under the [filter:APPNAME] section of the paste config) will be passed into the *__init__* method as kwargs.

A hypothetical configuration would look like:

[filter:analytics] redis_host = 127.0.0.1 paste.filter_factory = nova.api.analytics:Analytics.factory

which would result in a call to the *Analytics* class as

import nova.api.analytics analytics.Analytics(app_from_paste, redis_host='127.0.0.1')

You could of course re-implement the *factory* method in subclasses, but using the kwarg passing it shouldn't be necessary.

process_request(*req*)
Called on each request.

If this returns None, the next application down the stack will be executed. If it returns a response then that response will be returned and execution will stop here.

process_response(*response*)
Do whatever you'd like to the response.

class **Request**(*environ*, *charset=None*, *unicode_errors=None*, *decode_param_names=None*, ***kw*)
Bases: webob.request.Request

class **Router**(*mapper*)
Bases: object

WSGI middleware that maps incoming requests to WSGI apps.

class **Server**(*name*, *app*, *host='0.0.0.0'*, *port=0*, *pool_size=None*, *protocol=<class eventlet.wsgi.HttpProtocol at 0x7f858d477b48>*, *backlog=128*, *use_ssl=False*, *max_url_len=None*)
Bases: object

Server class to manage a WSGI server, serving a WSGI application.

default_pool_size = 1000

reset()
Reset server greenpool size to default.

Returns None

start()
Start serving a WSGI application.

Returns None

stop()
Stop this server.

This is not a very nice action, as currently the method by which a server is stopped is by killing its eventlet.

Returns None

wait()

Block, until the server has stopped.

Waits on the server's eventlet to finish, then returns.

Returns None

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

a

nova.api.auth, 117
nova.api.compute_req_id, 118
nova.api.ec2.apirequest, 118
nova.api.ec2.cloud, 118
nova.api.ec2.ec2utils, 120
nova.api.ec2.faults, 122
nova.api.ec2.inst_state, 122
nova.api.manager, 122
nova.api.metadata.base, 122
nova.api.metadata.handler, 123
nova.api.metadata.password, 123
nova.api.metadata.vendordata_json, 123
nova.api.openstack.api_version_request, 124
nova.api.openstack.auth, 124
nova.api.openstack.common, 124
nova.api.openstack.compute.consoles, 126
nova.api.openstack.compute.contrib.admin_actions, 127
nova.api.openstack.compute.contrib.agents, 127
nova.api.openstack.compute.contrib.aggregates, 128
nova.api.openstack.compute.contrib.assisted_volume_snapshots, 128
nova.api.openstack.compute.contrib.attach_interfaces, 129
nova.api.openstack.compute.contrib.availability_zone, 130
nova.api.openstack.compute.contrib.baremetal_ext_status, 130
nova.api.openstack.compute.contrib.baremetal_nodes, 130
nova.api.openstack.compute.contrib.block_device_mapping_v2_boot, 131
nova.api.openstack.compute.contrib.cell_capacities, 131
nova.api.openstack.compute.contrib.cells, 131
nova.api.openstack.compute.contrib.certificates, 132
nova.api.openstack.compute.contrib.cloudpipe, 133
nova.api.openstack.compute.contrib.cloudpipe_update, 133
nova.api.openstack.compute.contrib.config_drive, 134
nova.api.openstack.compute.contrib.console_auth_tok, 134
nova.api.openstack.compute.contrib.console_output, 135
nova.api.openstack.compute.contrib.consoles, 135
nova.api.openstack.compute.contrib.createserverext, 136
nova.api.openstack.compute.contrib.deferred_delete, 136
nova.api.openstack.compute.contrib.disk_config, 136
nova.api.openstack.compute.contrib.evacuate, 137
nova.api.openstack.compute.contrib.extended_availab, 138
nova.api.openstack.compute.contrib.extended_evacuat, 138
nova.api.openstack.compute.contrib.extended_floatin, 138
nova.api.openstack.compute.contrib.extended_hypervi, 139
nova.api.openstack.compute.contrib.extended_ips, 139
nova.api.openstack.compute.contrib.extended_ips_mac, 139
nova.api.openstack.compute.contrib.extended_network, 140
nova.api.openstack.compute.contrib.extended_quotas, 140
nova.api.openstack.compute.contrib.extended_rescue_, 140
nova.api.openstack.compute.contrib.extended_server_, 141
nova.api.openstack.compute.contrib.extended_service, 141

nova.api.openstack.compute.contrib.extended_server_attributes, 141
nova.api.openstack.compute.contrib.extended_status, 142
nova.api.openstack.compute.contrib.extended_virtual_interfaces_net, 142
nova.api.openstack.compute.contrib.extended_volumes, 142
nova.api.openstack.compute.contrib.fixed_ips, 143
nova.api.openstack.compute.contrib.flavor_access, 143
nova.api.openstack.compute.contrib.flavor_disabled, 144
nova.api.openstack.compute.contrib.flavor_rxtx, 145
nova.api.openstack.compute.contrib.flavor_swap, 145
nova.api.openstack.compute.contrib.flavorextradata, 146
nova.api.openstack.compute.contrib.flavorextraspecs, 146
nova.api.openstack.compute.contrib.flavormanage, 147
nova.api.openstack.compute.contrib.floating_ip_dns, 147
nova.api.openstack.compute.contrib.floating_ip_pools, 148
nova.api.openstack.compute.contrib.floating_ips, 148
nova.api.openstack.compute.contrib.floating_ips_bulk, 149
nova.api.openstack.compute.contrib.fping, 149
nova.api.openstack.compute.contrib.hide_server_addresses, 150
nova.api.openstack.compute.contrib.hosts, 150
nova.api.openstack.compute.contrib.hypervisor_status, 152
nova.api.openstack.compute.contrib.hypervisors, 152
nova.api.openstack.compute.contrib.image_size, 153
nova.api.openstack.compute.contrib.instance_actions, 153
nova.api.openstack.compute.contrib.instance_usage_audit_log, 154
nova.api.openstack.compute.contrib.keypairs, 154
nova.api.openstack.compute.contrib.migrations, 155
nova.api.openstack.compute.contrib.multinic, 155
nova.api.openstack.compute.contrib.multiple_create, 156
nova.api.openstack.compute.contrib.networks_associate, 156
nova.api.openstack.compute.contrib.os_networks, 156
nova.api.openstack.compute.contrib.os_tenant_networks, 157
nova.api.openstack.compute.contrib.preserve_ephemeral_rebuild, 158
nova.api.openstack.compute.contrib.quota_classes, 158
nova.api.openstack.compute.contrib.quotas, 158
nova.api.openstack.compute.contrib.rescue, 159
nova.api.openstack.compute.contrib.scheduler_hints, 159
nova.api.openstack.compute.contrib.security_group_default_rules, 160
nova.api.openstack.compute.contrib.security_groups, 160
nova.api.openstack.compute.contrib.server_diagnostics, 161
nova.api.openstack.compute.contrib.server_external_events, 162
nova.api.openstack.compute.contrib.server_group_quotas, 162
nova.api.openstack.compute.contrib.server_groups, 163
nova.api.openstack.compute.contrib.server_list_multi_create_ext, 163
nova.api.openstack.compute.contrib.server_password, 163
nova.api.openstack.compute.contrib.server_sort_keys, 164
nova.api.openstack.compute.contrib.server_start_stop, 164
nova.api.openstack.compute.contrib.server_usage, 165
nova.api.openstack.compute.contrib.services, 165
nova.api.openstack.compute.contrib.shelve, 165
nova.api.openstack.compute.contrib.simple_tenant_usage, 166
nova.api.openstack.compute.contrib.used_limits, 166
nova.api.openstack.compute.contrib.used_limits_for_admin, 167
nova.api.openstack.compute.contrib.user_data, 167
nova.api.openstack.compute.contrib.user_quotas, 167

nova.api.openstack.compute.contrib.virtual_interfaces, 168
nova.api.openstack.compute.contrib.volume_attachments, 168
nova.api.openstack.compute.contrib.volumes, 168
nova.api.openstack.compute.extensions, 170
nova.api.openstack.compute.flavors, 170
nova.api.openstack.compute.image_metadata, 170
nova.api.openstack.compute.images, 171
nova.api.openstack.compute.ips, 171
nova.api.openstack.compute.limits, 172
nova.api.openstack.compute.plugins.v3.access, 173
nova.api.openstack.compute.plugins.v3.admin_actions, 174
nova.api.openstack.compute.plugins.v3.admin_password, 174
nova.api.openstack.compute.plugins.v3.agents, 175
nova.api.openstack.compute.plugins.v3.aggregates, 176
nova.api.openstack.compute.plugins.v3.assisted_volume_snapshots, 176
nova.api.openstack.compute.plugins.v3.attach_interfaces, 177
nova.api.openstack.compute.plugins.v3.availability_zone, 177
nova.api.openstack.compute.plugins.v3.baremetal_nodes, 178
nova.api.openstack.compute.plugins.v3.block_device_mapping, 179
nova.api.openstack.compute.plugins.v3.block_device_mapping_v1, 179
nova.api.openstack.compute.plugins.v3.cells, 179
nova.api.openstack.compute.plugins.v3.certificates, 180
nova.api.openstack.compute.plugins.v3.cloudpipe, 181
nova.api.openstack.compute.plugins.v3.config_drive, 182
nova.api.openstack.compute.plugins.v3.console_auth_tokens, 182
nova.api.openstack.compute.plugins.v3.console_output, 183
nova.api.openstack.compute.plugins.v3.consoles, 183
nova.api.openstack.compute.plugins.v3.create_backup, 184
nova.api.openstack.compute.plugins.v3.deferred_delete, 184
nova.api.openstack.compute.plugins.v3.disk_config, 184
nova.api.openstack.compute.plugins.v3.evacuate, 185
nova.api.openstack.compute.plugins.v3.extended_availability_zone, 186
nova.api.openstack.compute.plugins.v3.extended_server_attributes, 186
nova.api.openstack.compute.plugins.v3.extended_status, 187
nova.api.openstack.compute.plugins.v3.extended_volumes, 187
nova.api.openstack.compute.plugins.v3.extension_info, 188
nova.api.openstack.compute.plugins.v3.fixed_ips, 188
nova.api.openstack.compute.plugins.v3.flavor_access, 189
nova.api.openstack.compute.plugins.v3.flavor_manage, 189
nova.api.openstack.compute.plugins.v3.flavor_rxtx, 190
nova.api.openstack.compute.plugins.v3.flavors, 190
nova.api.openstack.compute.plugins.v3.flavors_extra_specs, 191
nova.api.openstack.compute.plugins.v3.floating_ip_dns, 192
nova.api.openstack.compute.plugins.v3.floating_ip_pools, 192
nova.api.openstack.compute.plugins.v3.floating_ips, 193
nova.api.openstack.compute.plugins.v3.floating_ips_bulk, 194
nova.api.openstack.compute.plugins.v3.fping, 194
nova.api.openstack.compute.plugins.v3.hide_server_addresses, 195
nova.api.openstack.compute.plugins.v3.hosts, 195
nova.api.openstack.compute.plugins.v3.hypervisors, 197
nova.api.openstack.compute.plugins.v3.image_metadata, 198
nova.api.openstack.compute.plugins.v3.image_size, 198
nova.api.openstack.compute.plugins.v3.images, 199
nova.api.openstack.compute.plugins.v3.instance_actions, 199
nova.api.openstack.compute.plugins.v3.instance_usage_audit_log, 200
nova.api.openstack.compute.plugins.v3.ips, 200

nova.api.openstack.compute.plugins.v3.keypairs, 201
nova.api.openstack.compute.plugins.v3.limits, 202
nova.api.openstack.compute.plugins.v3.lock_server, 202
nova.api.openstack.compute.plugins.v3.migrate_server, 203
nova.api.openstack.compute.plugins.v3.migrations, 203
nova.api.openstack.compute.plugins.v3.multinic, 204
nova.api.openstack.compute.plugins.v3.multiple_create, 204
nova.api.openstack.compute.plugins.v3.networks, 204
nova.api.openstack.compute.plugins.v3.networks_associate, 205
nova.api.openstack.compute.plugins.v3.pause_server, 205
nova.api.openstack.compute.plugins.v3.pci, 206
nova.api.openstack.compute.plugins.v3.personality, 206
nova.api.openstack.compute.plugins.v3.preserve_ephemeral_rebuild, 207
nova.api.openstack.compute.plugins.v3.quota_classes, 207
nova.api.openstack.compute.plugins.v3.quotas, 208
nova.api.openstack.compute.plugins.v3.remote_consoles, 208
nova.api.openstack.compute.plugins.v3.rescue, 209
nova.api.openstack.compute.plugins.v3.scheduler_hints, 209
nova.api.openstack.compute.plugins.v3.security_group_default_rules, 210
nova.api.openstack.compute.plugins.v3.security_groups, 210
nova.api.openstack.compute.plugins.v3.server_diagnostics, 212
nova.api.openstack.compute.plugins.v3.server_external_events, 212
nova.api.openstack.compute.plugins.v3.server_groups, 213
nova.api.openstack.compute.plugins.v3.server_metadata, 213
nova.api.openstack.compute.plugins.v3.server_password, 214
nova.api.openstack.compute.plugins.v3.server_usage, 214
nova.api.openstack.compute.plugins.v3.servers, 215
nova.api.openstack.compute.plugins.v3.services, 216
nova.api.openstack.compute.plugins.v3.shelve, 216
nova.api.openstack.compute.plugins.v3.simple_tenant_usage, 217
nova.api.openstack.compute.plugins.v3.suspend_server, 217
nova.api.openstack.compute.plugins.v3.tenant_networks, 218
nova.api.openstack.compute.plugins.v3.used_limits, 218
nova.api.openstack.compute.plugins.v3.user_data, 219
nova.api.openstack.compute.plugins.v3.versions, 219
nova.api.openstack.compute.plugins.v3.virtual_interfaces, 220
nova.api.openstack.compute.plugins.v3.volumes, 220
nova.api.openstack.compute.schemas.v3.access_ips, 223
nova.api.openstack.compute.schemas.v3.admin_password, 223
nova.api.openstack.compute.schemas.v3.agents, 223
nova.api.openstack.compute.schemas.v3.aggregates, 223
nova.api.openstack.compute.schemas.v3.assisted_volume_snapshots, 223
nova.api.openstack.compute.schemas.v3.attach_interfaces, 223
nova.api.openstack.compute.schemas.v3.availability_zone, 223
nova.api.openstack.compute.schemas.v3.block_device_mapping, 223
nova.api.openstack.compute.schemas.v3.block_device_mapping_v1, 223
nova.api.openstack.compute.schemas.v3.cells, 223
nova.api.openstack.compute.schemas.v3.cloudpipe, 223
nova.api.openstack.compute.schemas.v3.config_drive, 223
nova.api.openstack.compute.schemas.v3.console_output, 223
nova.api.openstack.compute.schemas.v3.create_backup, 223
nova.api.openstack.compute.schemas.v3.disk_config, 223
nova.api.openstack.compute.schemas.v3.evacuate, 223
nova.api.openstack.compute.schemas.v3.fixed_ips, 223

nova.api.openstack.compute.schemas.v3.flavor_access, 223
nova.api.openstack.compute.schemas.v3.flavor_manage, 223
nova.api.openstack.compute.schemas.v3.flavors_extra_specs, 223
nova.api.openstack.compute.schemas.v3.floating_ip_dns, 223
nova.api.openstack.compute.schemas.v3.floating_ip_pools, 223
nova.api.openstack.compute.schemas.v3.floating_ips, 223
nova.api.openstack.compute.schemas.v3.hosts, 223
nova.api.openstack.compute.schemas.v3.image_metadata, 223
nova.api.openstack.compute.schemas.v3.keypairs, 223
nova.api.openstack.compute.schemas.v3.migrate_server, 223
nova.api.openstack.compute.schemas.v3.multinic, 223
nova.api.openstack.compute.schemas.v3.multiple_create, 223
nova.api.openstack.compute.schemas.v3.networks, 223
nova.api.openstack.compute.schemas.v3.networks_associate, 223
nova.api.openstack.compute.schemas.v3.personality, 223
nova.api.openstack.compute.schemas.v3.preserve_ephemeral_rebuild, 223
nova.api.openstack.compute.schemas.v3.quota_classes, 223
nova.api.openstack.compute.schemas.v3.quota_sets, 223
nova.api.openstack.compute.schemas.v3.remote_consoles, 223
nova.api.openstack.compute.schemas.v3.rescue, 223
nova.api.openstack.compute.schemas.v3.reset_server_state, 223
nova.api.openstack.compute.schemas.v3.scheduler_hints, 223
nova.api.openstack.compute.schemas.v3.security_groups, 223
nova.api.openstack.compute.schemas.v3.server_external_events, 223
nova.api.openstack.compute.schemas.v3.server_groups, 223
nova.api.openstack.compute.schemas.v3.server_metadata, 223
nova.api.openstack.compute.schemas.v3.servers, 223
nova.api.openstack.compute.schemas.v3.services, 223
nova.api.openstack.compute.schemas.v3.tenant_networks, 223
nova.api.openstack.compute.schemas.v3.user_data, 223
nova.api.openstack.compute.schemas.v3.volumes, 223
nova.api.openstack.compute.server_metadata, 223
nova.api.openstack.compute.servers, 224
nova.api.openstack.compute.versions, 224
nova.api.openstack.compute.views.addresses, 225
nova.api.openstack.compute.views.flavors, 225
nova.api.openstack.compute.views.images, 226
nova.api.openstack.compute.views.limits, 226
nova.api.openstack.compute.views.servers, 226
nova.api.openstack.compute.views.versions, 227
nova.api.openstack.extensions, 227
nova.api.openstack.urlmap, 229
nova.api.openstack.versioned_method, 230
nova.api.openstack.wsgi, 230
nova.api.opts, 234
nova.api.sizelimit, 234
nova.api.validation.parameter_types, 234
nova.api.validation.validators, 234
nova.api.validator, 234
nova.availability_zones, 235

b

nova.baserpc, 235
nova.block_device, 236

c

nova.cells.driver, 237
nova.cells.filters.different_cell, 237
nova.cells.filters.image_properties, 238
nova.cells.filters.target_cell, 238
nova.cells.manager, 238
nova.cells.messaging, 241
nova.cells.opts, 244
nova.cells.rpc_driver, 245
nova.cells.rpcapi, 246
nova.cells.scheduler, 250
nova.cells.state, 250
nova.cells.utils, 251
nova.cells.weights.mute_child, 252

nova.cells.weights.ram_by_instance_type, 252
nova.cells.weights.weight_offset, 252
nova.cert.manager, 253
nova.cert.rpcapi, 253
nova.cloudpipe.pipelib, 254
nova.cmd.all, 254
nova.cmd.api, 254
nova.cmd.api_ec2, 254
nova.cmd.api_metadata, 255
nova.cmd.api_os_compute, 255
nova.cmd.baseproxy, 255
nova.cmd.cells, 255
nova.cmd.cert, 255
nova.cmd.compute, 255
nova.cmd.conductor, 255
nova.cmd.console, 255
nova.cmd.consoleauth, 256
nova.cmd.dhcpbridge, 256
nova.cmd.idmapshift, 256
nova.cmd.manage, 257
nova.cmd.network, 261
nova.cmd.novnc, 261
nova.cmd.novncproxy, 261
nova.cmd.objectstore, 261
nova.cmd.scheduler, 261
nova.cmd.serialproxy, 262
nova.cmd.spicehtml5proxy, 262
nova.cmd.xvpvncproxy, 262
nova.compute.api, 262
nova.compute.arch, 269
nova.compute.build_results, 269
nova.compute.cells_api, 269
nova.compute.claims, 272
nova.compute.cpumodel, 273
nova.compute.flavors, 273
nova.compute.hv_type, 273
nova.compute.instance_actions, 274
nova.compute.manager, 274
nova.compute.monitors.base, 281
nova.compute.monitors.cpu.virt_driver, 282
nova.compute.opts, 282
nova.compute.power_state, 282
nova.compute.resource_tracker, 282
nova.compute.resources.base, 283
nova.compute.resources.vcpu, 284
nova.compute.rpcapi, 284
nova.compute.stats, 291
nova.compute.task_states, 292
nova.compute.utils, 292
nova.compute.vm_mode, 294
nova.compute.vm_states, 294
nova.conductor.api, 295
nova.conductor.manager, 296
nova.conductor.rpcapi, 298
nova.conductor.tasks.live_migrate, 302
nova.config, 302
nova.console.api, 302
nova.console.fake, 302
nova.console.manager, 303
nova.console.rpcapi, 303
nova.console.serial, 304
nova.console.type, 304
nova.console.websocketproxy, 304
nova.console.xvp, 305
nova.consoleauth.manager, 305
nova.consoleauth.rpcapi, 305
nova.context, 306
nova.crypto, 307

d

nova.db.api, 308
nova.db.base, 323
nova.db.migration, 323
nova.db.sqlalchemy.api, 324
nova.db.sqlalchemy.api_migrations.migrate_repo.vers 336
nova.db.sqlalchemy.api_migrations.migrate_repo.vers 336
nova.db.sqlalchemy.api_migrations.migrate_repo.vers 336
nova.db.sqlalchemy.api_models, 336
nova.db.sqlalchemy.migrate_repo.manage, 337
nova.db.sqlalchemy.migrate_repo.versions.216_havana 337
nova.db.sqlalchemy.migrate_repo.versions.217_placeh 337
nova.db.sqlalchemy.migrate_repo.versions.218_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.219_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.220_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.221_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.222_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.223_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.224_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.225_placeh 338
nova.db.sqlalchemy.migrate_repo.versions.226_placeh 338

nova.db.sqlalchemy.migrate_repo.versions.227_drop_migrate_version_table,
nova.db.sqlalchemy.migrate_repo.versions.228_add_migration_progress,
nova.db.sqlalchemy.migrate_repo.versions.229_add_uefi_image_property,
nova.db.sqlalchemy.migrate_repo.versions.230_add_instance_extra_table,
nova.db.sqlalchemy.migrate_repo.versions.231_add_pci_requests_to_instance_extra_table,
nova.db.sqlalchemy.migrate_repo.versions.232_drop_dump_tables,
nova.db.sqlalchemy.migrate_repo.versions.233_add_stats_to_compute_nodes,
nova.db.sqlalchemy.migrate_repo.versions.234_add_uefi_boot_mode_to_instance_extra,
nova.db.sqlalchemy.migrate_repo.versions.235_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.236_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.237_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.238_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.239_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.240_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.241_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.242_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.243_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.244_increase_length_volume_usage_cache,
nova.db.sqlalchemy.migrate_repo.versions.245_add_mtu_and_dhcp_server,
nova.db.sqlalchemy.migrate_repo.versions.246_add_compute_node_id_fk,
nova.db.sqlalchemy.migrate_repo.versions.247_nullable_mismatch,
nova.db.sqlalchemy.migrate_repo.versions.248_add_expire_reservations_index,
nova.db.sqlalchemy.migrate_repo.versions.249_remove_duplicate_index,
nova.db.sqlalchemy.migrate_repo.versions.250_remove_instances_uuid_unique_constraint,
nova.db.sqlalchemy.migrate_repo.versions.251_add_numa_topology_to_compute_nodes,
nova.db.sqlalchemy.migrate_repo.versions.252_add_instance_extra_table,
nova.db.sqlalchemy.migrate_repo.versions.253_add_pci_requests_to_instance_extra_table,
nova.db.sqlalchemy.migrate_repo.versions.254_add_request_id,
nova.db.sqlalchemy.migrate_repo.versions.255_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.256_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.257_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.258_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.259_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.260_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.261_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.262_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.263_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.264_placeholder,
nova.db.sqlalchemy.migrate_repo.versions.265_remove_cn_pci_stats,
nova.db.sqlalchemy.migrate_repo.versions.266_add_instance_extra_migration_context,
nova.db.sqlalchemy.migrate_repo.versions.267_instance_uuid_non_nullable,
nova.db.sqlalchemy.migrate_repo.versions.268_add_host_mapping_status,
nova.db.sqlalchemy.migrate_repo.versions.269_add_numa_topology_to_instance_extra,
nova.db.sqlalchemy.migrate_repo.versions.270_flavor_description,
nova.db.sqlalchemy.migrate_repo.versions.271_change_instance_uuid_to_string,
nova.db.sqlalchemy.migrate_repo.versions.272_add_keypair_type,
nova.db.sqlalchemy.migrate_repo.versions.273_sqlite_drop_unique_fk,
nova.db.sqlalchemy.migrate_repo.versions.274_update_instances_stale_on_mysql,
nova.db.sqlalchemy.migrate_repo.versions.275_add_keypair_type_index,
nova.db.sqlalchemy.migrate_repo.versions.276_vcpu_model,
nova.db.sqlalchemy.migrate_repo.versions.277_add_flavor_extra_specs_table,
nova.db.sqlalchemy.migrate_repo.versions.278_remove_duplicate_fixed_ips_virtual_interface_id_idx,
nova.db.sqlalchemy.migrate_repo.versions.279_fix_uuid_columns_for_mysql,
nova.db.sqlalchemy.migrate_repo.versions.280_add_numa_topology_to_compute_nodes,

nova.db.sqlalchemy.migrate_repo.versions.281_image_id, 345
nova.db.sqlalchemy.migrate_repo.versions.282_placeholder, 345
nova.db.sqlalchemy.migrate_repo.versions.283_placeholder, 345
nova.db.sqlalchemy.migrate_repo.versions.284_placeholder, 345
nova.db.sqlalchemy.migrate_repo.versions.285_placeholder, 346
nova.db.sqlalchemy.migrate_repo.versions.286_placeholder, 346
nova.db.sqlalchemy.migrate_repo.versions.287_placeholder, 346
nova.db.sqlalchemy.migrate_repo.versions.288_placeholder, 346
nova.db.sqlalchemy.migrate_repo.versions.289_placeholder, 346
nova.db.sqlalchemy.migrate_repo.versions.290_placeholder, 346
nova.db.sqlalchemy.migrate_repo.versions.291_enforce_flavors_migrated, 346
nova.db.sqlalchemy.migrate_repo.versions.292_drop_nova_volumes_tables, 346
nova.db.sqlalchemy.migrate_repo.versions.293_add_migration_type, 346
nova.db.sqlalchemy.migrate_repo.versions.294_add_service_heartbeat, 347
nova.db.sqlalchemy.migrate_repo.versions.295_add_virtual_interfaces_uuid_index, 347
nova.db.sqlalchemy.migrate_repo.versions.296_add_missing_db2_fkeys, 347
nova.db.sqlalchemy.migration, 347
nova.db.sqlalchemy.models, 349
nova.db.sqlalchemy.types, 371
nova.db.sqlalchemy.utils, 371
nova.debugger, 372

e

nova.exception, 372

f

nova.filters, 403

h

nova.hacking.checks, 403
nova.hooks, 406

i

nova.i18n, 407
nova.image.api, 407
nova.image.download.base, 409
nova.image.download.file, 409
nova.image.glance, 409
nova.ipv6.account_identifier, 411
nova.ipv6.api, 411
nova.ipv6.rfc2462, 411

k

nova.keymgr.barbican, 411
nova.keymgr.conf_key_mgr, 413
nova.keymgr.key, 413
nova.keymgr.key_mgr, 414
nova.keymgr.mock_key_mgr, 415
nova.keymgr.not_implemented_key_mgr, 415
nova.keymgr.single_key_mgr, 416

l

nova.loadables, 416

m

nova.manager, 416

n

nova.netconf, 417
nova.network.api, 417
nova.network.base_api, 420
nova.network.dns_driver, 423
nova.network.driver, 423
nova.network.floating_ips, 424
nova.network.l3, 425
nova.network.ldapdns, 426
nova.network.linux_net, 427
nova.network.manager, 431
nova.network.minidns, 435
nova.network.model, 436
nova.network.neutronv2.api, 438
nova.network.neutronv2.constants, 441
nova.network.noop_dns_driver, 441
nova.network.opts, 441
nova.network.rpcapi, 442
nova.network.security_group.neutron_driver, 444
nova.network.security_group.openstack_driver, 445
nova.network.security_group.security_group_base, 445
nova.notifications, 446

o

nova.objects.agent, 447
nova.objects.aggregate, 448
nova.objects.bandwidth_usage, 448
nova.objects.base, 449
nova.objects.block_device, 452
nova.objects.cell_mapping, 454
nova.objects.compute_node, 454

nova.objects.dns_domain, 456
nova.objects.ec2, 457
nova.objects.external_event, 458
nova.objects.fields, 459
nova.objects.fixed_ip, 465
nova.objects.flavor, 466
nova.objects.floating_ip, 467
nova.objects.host_mapping, 469
nova.objects.hv_spec, 469
nova.objects.image_meta, 469
nova.objects.instance, 472
nova.objects.instance_action, 476
nova.objects.instance_fault, 477
nova.objects.instance_group, 478
nova.objects.instance_info_cache, 479
nova.objects.instance_mapping, 480
nova.objects.instance_numa_topology, 480
nova.objects.instance_pci_requests, 482
nova.objects.keypair, 482
nova.objects.migration, 483
nova.objects.monitor_metric, 484
nova.objects.network, 485
nova.objects.network_request, 486
nova.objects.numa, 487
nova.objects.pci_device, 488
nova.objects.pci_device_pool, 490
nova.objects.quotas, 491
nova.objects.security_group, 492
nova.objects.security_group_rule, 492
nova.objects.service, 493
nova.objects.tag, 494
nova.objects.task_log, 495
nova.objects.vcpu_model, 496
nova.objects.virt_cpu_topology, 496
nova.objects.virtual_interface, 497
nova.objectstore.s3server, 498
nova.openstack.common._i18n, 499
nova.openstack.common.cliutils, 499
nova.openstack.common.eventlet_backdoor, 501
nova.openstack.common.fileutils, 501
nova.openstack.common.imageutils, 502
nova.openstack.common.loopingcall, 502
nova.openstack.common.memorycache, 503
nova.openstack.common.periodic_task, 503
nova.openstack.common.policy, 504
nova.openstack.common.report.generators.conf, 508
nova.openstack.common.report.generators.process, 508
nova.openstack.common.report.generators.threading, 508
nova.openstack.common.report.generators.version, 509
nova.openstack.common.report.guru_meditation_report, 509
nova.openstack.common.report.models.base, 511
nova.openstack.common.report.models.conf, 512
nova.openstack.common.report.models.process, 512
nova.openstack.common.report.models.threading, 512
nova.openstack.common.report.models.version, 513
nova.openstack.common.report.models.with_default_v, 513
nova.openstack.common.report.report, 514
nova.openstack.common.report.utils, 516
nova.openstack.common.report.views.jinja_view, 516
nova.openstack.common.report.views.json.generic, 517
nova.openstack.common.report.views.text.generic, 517
nova.openstack.common.report.views.text.header, 518
nova.openstack.common.report.views.text.process, 519
nova.openstack.common.report.views.text.threading, 519
nova.openstack.common.report.views.xml.generic, 519
nova.openstack.common.service, 520
nova.openstack.common.sslutils, 521
nova.openstack.common.systemd, 521
nova.openstack.common.threadgroup, 522
nova.opts, 523

p

nova.paths, 523
nova.pci.device, 523
nova.pci.devspec, 523
nova.pci.manager, 524
nova.pci.request, 525
nova.pci.stats, 525
nova.pci.utils, 526
nova.pci.whitelist, 527
nova.policy, 527

q

nova.quota, 528

r

nova.rpc, 539

s

nova.safe_utils, 540

nova.scheduler.caching_scheduler, 540
nova.scheduler.chance, 540
nova.scheduler.client.query, 541
nova.scheduler.client.report, 541
nova.scheduler.driver, 542
nova.scheduler.filter_scheduler, 542
nova.scheduler.filters.affinity_filter, 542
nova.scheduler.filters.aggregate_image_properties_isolation, 543
nova.scheduler.filters.aggregate_instance_extra_specs, 543
nova.scheduler.filters.aggregate_multitenancy_isolation, 543
nova.scheduler.filters.all_hosts_filter, 544
nova.scheduler.filters.availability_zone_filter, 544
nova.scheduler.filters.compute_capabilities_filter, 544
nova.scheduler.filters.compute_filter, 544
nova.scheduler.filters.core_filter, 545
nova.scheduler.filters.disk_filter, 545
nova.scheduler.filters.exact_core_filter, 545
nova.scheduler.filters.exact_disk_filter, 545
nova.scheduler.filters.exact_ram_filter, 546
nova.scheduler.filters.extra_specs_ops, 546
nova.scheduler.filters.image_props_filter, 546
nova.scheduler.filters.io_ops_filter, 546
nova.scheduler.filters.isolated_hosts_filter, 546
nova.scheduler.filters.json_filter, 547
nova.scheduler.filters.metrics_filter, 547
nova.scheduler.filters.num_instances_filter, 547
nova.scheduler.filters.numa_topology_filter, 548
nova.scheduler.filters.pci_passthrough_filter, 548
nova.scheduler.filters.ram_filter, 548
nova.scheduler.filters.retry_filter, 548
nova.scheduler.filters.trusted_filter, 549
nova.scheduler.filters.type_filter, 550
nova.scheduler.filters.utils, 550
nova.scheduler.host_manager, 550
nova.scheduler.ironic_host_manager, 552
nova.scheduler.manager, 552
nova.scheduler.opts, 553
nova.scheduler.rpcapi, 553
nova.scheduler.scheduler_options, 554
nova.scheduler.utils, 555
nova.scheduler.weights.io_ops, 556
nova.scheduler.weights.metrics, 556
nova.scheduler.weights.ram, 556
nova.service, 556
nova.servicegroup.api, 558
nova.servicegroup.drivers.base, 558
nova.servicegroup.drivers.db, 558
nova.servicegroup.drivers.mc, 559
nova.servicegroup.drivers.zk, 559
nova.storage.linuxscsi, 559

t

nova.test, 559

u

nova.utils, 561

v

nova.version, 566
nova.virt.block_device, 566
nova.virt.configdrive, 567
nova.virt.diagnostics, 568
nova.virt.disk.api, 568
nova.virt.disk.mount.api, 570
nova.virt.disk.mount.loop, 571
nova.virt.disk.mount.nbd, 571
nova.virt.disk.vfs.api, 572
nova.virt.disk.vfs.guestfs, 573
nova.virt.disk.vfs.localfs, 573
nova.virt.driver, 574
nova.virt.event, 591
nova.virt.fake, 592
nova.virt.firewall, 595
nova.virt.hardware, 597
nova.virt.hyperv.basevolumeutils, 599
nova.virt.hyperv.constants, 599
nova.virt.hyperv.driver, 600
nova.virt.hyperv.hostops, 601
nova.virt.hyperv.hostutils, 602
nova.virt.hyperv.hostutilsv2, 602
nova.virt.hyperv.imagecache, 602
nova.virt.hyperv.ioutils, 603
nova.virt.hyperv.livemigrationops, 603
nova.virt.hyperv.livemigrationutils, 603
nova.virt.hyperv.migrationops, 603
nova.virt.hyperv.networkutils, 604
nova.virt.hyperv.networkutilsv2, 604
nova.virt.hyperv.pathutils, 604

nova.virt.hyperv.rdpconsoleops, 605
nova.virt.hyperv.rdpconsoleutils, 605
nova.virt.hyperv.rdpconsoleutilsv2, 605
nova.virt.hyperv.snapshotops, 605
nova.virt.hyperv.utilsfactory, 606
nova.virt.hyperv.vhdutils, 606
nova.virt.hyperv.vhdutilsv2, 606
nova.virt.hyperv.vif, 607
nova.virt.hyperv.vmops, 607
nova.virt.hyperv.vmutils, 608
nova.virt.hyperv.vmutilsv2, 610
nova.virt.hyperv.volumeops, 610
nova.virt.hyperv.volumeutils, 611
nova.virt.hyperv.volumeutilsv2, 612
nova.virt.image.model, 612
nova.virt.imagecache, 613
nova.virt.images, 613
nova.virt.ironic.client_wrapper, 613
nova.virt.ironic.driver, 613
nova.virt.ironic.ironic_states, 618
nova.virt.ironic.patcher, 619
nova.virt.libvirt.blockinfo, 620
nova.virt.libvirt.compat, 622
nova.virt.libvirt.config, 622
nova.virt.libvirt.designer, 628
nova.virt.libvirt.dmcrypt, 629
nova.virt.libvirt.driver, 629
nova.virt.libvirt.firewall, 635
nova.virt.libvirt.guest, 635
nova.virt.libvirt.host, 638
nova.virt.libvirt.imagebackend, 640
nova.virt.libvirt.imagecache, 643
nova.virt.libvirt.lvm, 644
nova.virt.libvirt.quobyte, 645
nova.virt.libvirt.rbd_utils, 645
nova.virt.libvirt.remotefs, 646
nova.virt.libvirt.utils, 646
nova.virt.libvirt.vif, 650
nova.virt.libvirt.volume, 652
nova.virt.netutils, 655
nova.virt.opts, 655
nova.virt.storage_users, 655
nova.virt.virtapi, 656
nova.virt.vmwareapi.constants, 656
nova.virt.vmwareapi.driver, 656
nova.virt.vmwareapi.ds_util, 659
nova.virt.vmwareapi.error_util, 660
nova.virt.vmwareapi.host, 660
nova.virt.vmwareapi.imagecache, 660
nova.virt.vmwareapi.images, 661
nova.virt.vmwareapi.io_util, 662
nova.virt.vmwareapi.network_util, 663
nova.virt.vmwareapi.read_write_util, 663
nova.virt.vmwareapi.vif, 663
nova.virt.vmwareapi.vim_util, 663
nova.virt.vmwareapi.vm_util, 664
nova.virt.vmwareapi.vmops, 668
nova.virt.vmwareapi.volumeops, 671
nova.virt.volumeutils, 671
nova.virt.watchdog_actions, 671
nova.virt.xenapi.agent, 671
nova.virt.xenapi.client.objects, 672
nova.virt.xenapi.client.session, 674
nova.virt.xenapi.driver, 674
nova.virt.xenapi.fake, 679
nova.virt.xenapi.firewall, 681
nova.virt.xenapi.host, 682
nova.virt.xenapi.image.bittorrent, 682
nova.virt.xenapi.image.glance, 682
nova.virt.xenapi.image.utils, 683
nova.virt.xenapi.image.vdi_through_dev, 683
nova.virt.xenapi.network_utils, 684
nova.virt.xenapi.pool, 684
nova.virt.xenapi.pool_states, 684
nova.virt.xenapi.vif, 684
nova.virt.xenapi.vm_utils, 685
nova.virt.xenapi.vmops, 688
nova.virt.xenapi.volume_utils, 692
nova.virt.xenapi.volumeops, 693
nova.vnc.xvp_proxy, 693
nova.volume.cinder, 694
nova.volume.encryptors.base, 695
nova.volume.encryptors.cryptsetup, 695
nova.volume.encryptors.luks, 695
nova.volume.encryptors.nop, 696

W

nova.weights, 696
nova.wsgi, 697