
Nova Documentation

Release 12.0.0.0b2.dev87

OpenStack Foundation

July 07, 2015

CONTENTS

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	84
3.5	Advanced testing and guides	86
3.6	Man Pages	97
3.7	Module Reference	116
4	Indices and tables	119

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

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-getVNCCConsole'`, `'os-getSPICEConsole'`, `'os-getSerialConsole'` and `'os-getRDPCConsole'` 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/52415800-8b69-11e0-9b19-734f6f006e54"
          },
          {
            "rel": "bookmark",
            "href": "http://servers.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f6f006e54"
          }
        ]
      },
      "flavor" : {
        "id": "52415800-8b69-11e0-9b19-734f216543fd",
        "links": [
          {
            "rel": "self",
            "href": "http://servers.api.openstack.org/v2/1234/flavors/52415800-8b69-11e0-9b19-734f216543fd"
          },
          {
            "rel": "bookmark",
            "href": "http://servers.api.openstack.org/1234/flavors/52415800-8b69-11e0-9b19-734f216543fd"
          }
        ]
      },
      "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/52415800-8b69-11e0-9b19-708140000000"
    },
    {
        "rel": "bookmark",
        "href": "http://servers.api.openstack.org/1234/servers/52415800-8b69-11e0-9b19-708140000000"
    }
],
"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": "e4d909c290d0fblca068ffa222cb0",
    "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/52415800-8b69-11e0-9b19-734f6f007777"
      }
    ]
  }
}

```

```
    },
    {
      "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/52415800-8b69-11e0-9b19-734f5"
      },
      {
        "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
<code>maxTotalRAMSize</code>	51200	Maximum total amount of RAM (MB)
<code>maxServerMeta</code>	5	Maximum number of metadata items associated with a server.
<code>maxImageMeta</code>	5	Maximum number of metadata items associated with an image.
<code>maxPersonality</code>	5	The maximum number of file path/content pairs that can be supplied on server build.
<code>maxPersonality-Size</code>	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": "ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBpdCBtb3ZlcyBpbjBqdXN0IHN1Y2gg",
  "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": "52415800-8b69-11e0-9b19-734fcede0043",
    "name": "my-server",
    "links": [
      {
        "rel": "self",
        "href": "http://servers.api.openstack.org/v2/1234/servers/52415800-8b69-11e0-9b19-734fcede0043"
      },
      {
        "rel": "bookmark",
        "href": "http://servers.api.openstack.org/1234/servers/52415800-8b69-11e0-9b19-734fcede0043"
      }
    ]
  }
}
```

```

    }
  ]
}

```

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/52415800-8b69-11e0-9b19-734f5736d2a2"
      },
      {
        "rel" : "bookmark",
        "href" : "http://servers.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f5736d2a2"
      },
      {
        "rel" : "alternate",
        "type" : "application/vnd.openstack.image",
        "href" : "http://glance.api.openstack.org/1234/images/52415800-8b69-11e0-9b19-734f5736d2a2"
      }
    ]
  }
}

```

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 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-8000-000000000000",
      "rel": "next"
    }
  ]
}

```

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

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/52415800-8b69-11e0-9b19-734f"
        }
      ],
      "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).

HYPERVERSOR 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

<i>Feature</i>	<i>Status</i>	Hyper-V	Ironic	Libvirt KVM (ppc64)	Libvirt KVM (s390x)	Libvirt KVM (x86_64)
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)	Libvirt KVM (x86)
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 cumulative 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. Therefore 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 connectivity 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

3.1 Introduction

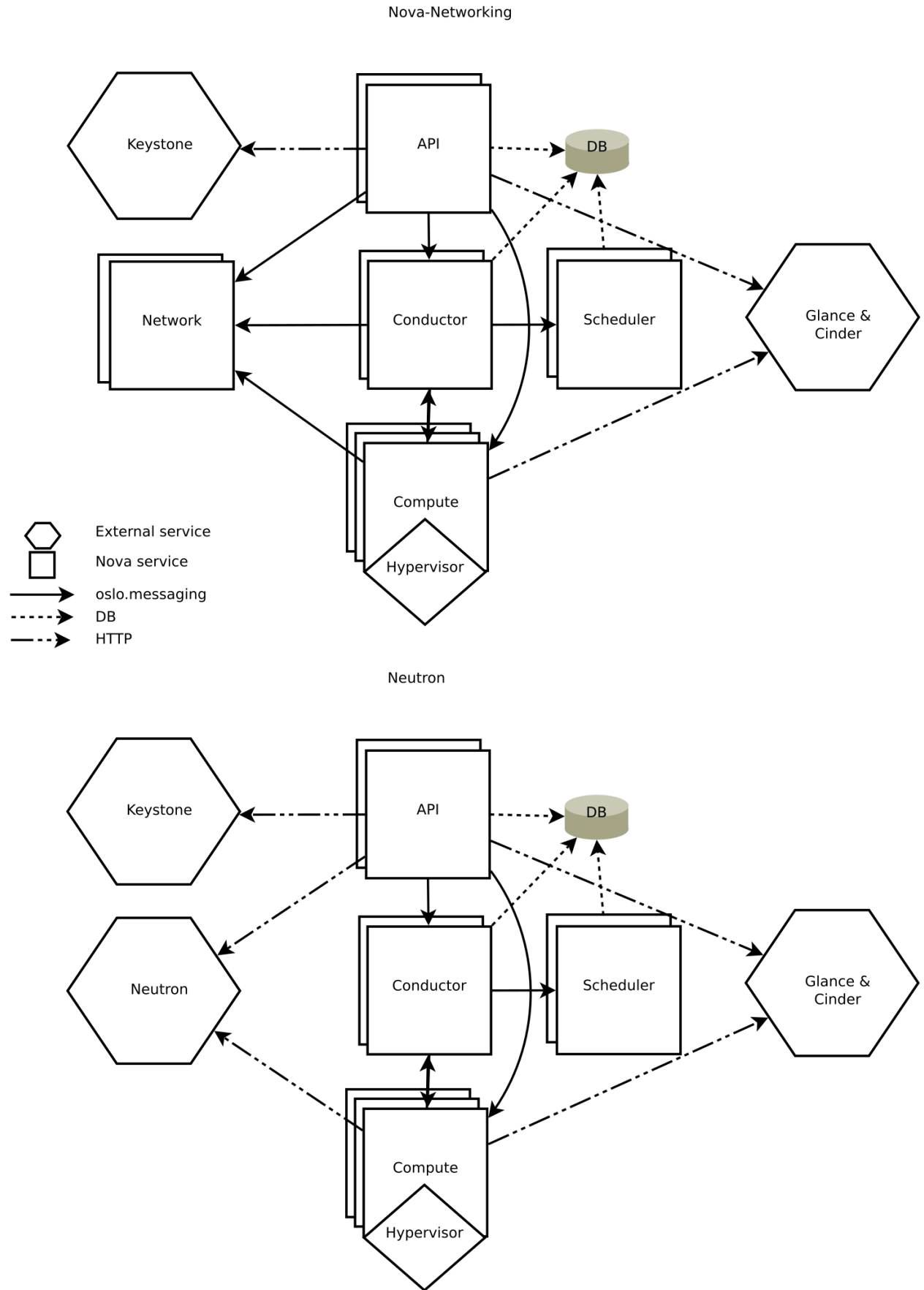
3.1.1 Nova System Architecture

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.



- 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 than 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 will shrink over time.

Driver Parity

Our goal for the Nova API is 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 were 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.

It's 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 to 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 providers provide 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 with 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 axes 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 as 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-devel
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

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.env`, `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 defined 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/cgiit/openstack/nova/tree/nova/nova/api/openstack/com

```

```

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

    name = "BasicTest"
    alias = ALIAS
    version = 1

    # Both get_resources and get_controller_extensions must always
    # be defined 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 be 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_microversion`

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_ids)”, “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 ag
* aggregate-add-host    <id> <host> Add the host to the specified ag
* aggregate-remove-host <id> <host> Remove the specified host from th
* aggregate-set-metadata <id> <key=value> [<key=value> ...] Update the metadata associated w
* 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 mainten
```

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 in-
stance_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.

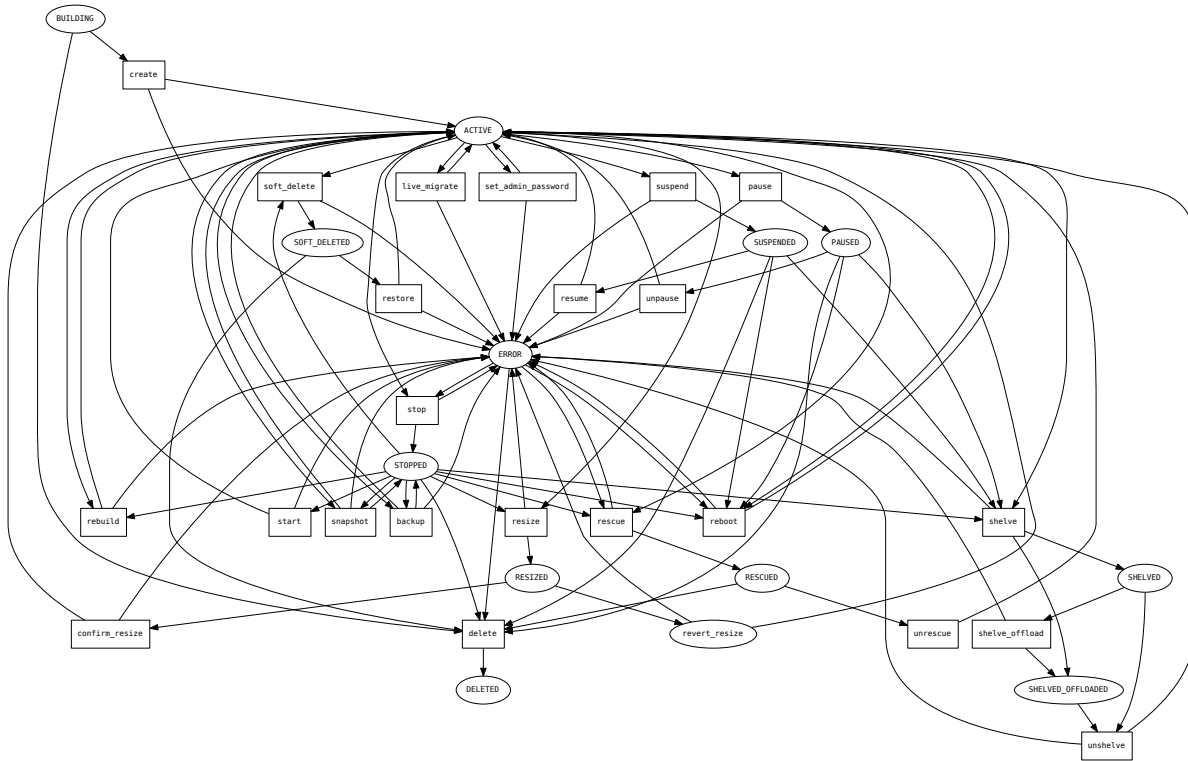
The Diablo release contained a thread-pooling implementation that did not block, but this implementation resulted in a bug and was removed.

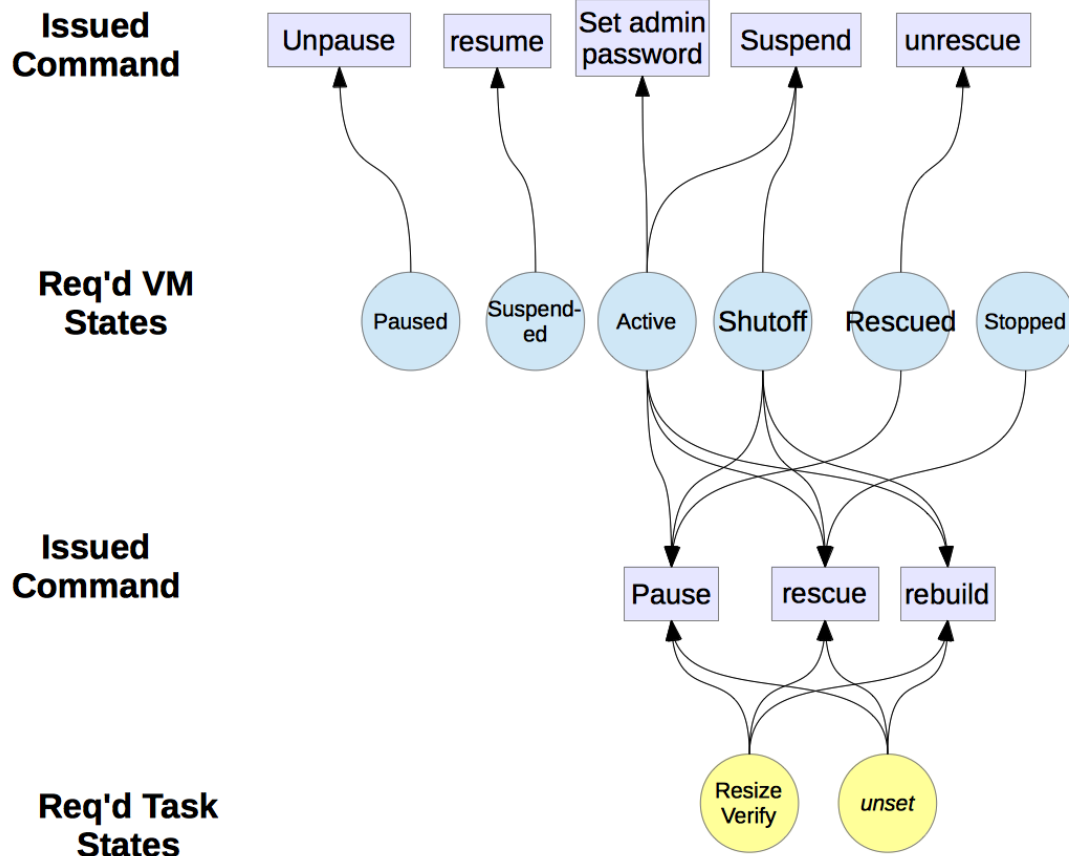
See this [mailing list thread](#) for a discussion of this issue, including a discussion of the [impact on performance](#).

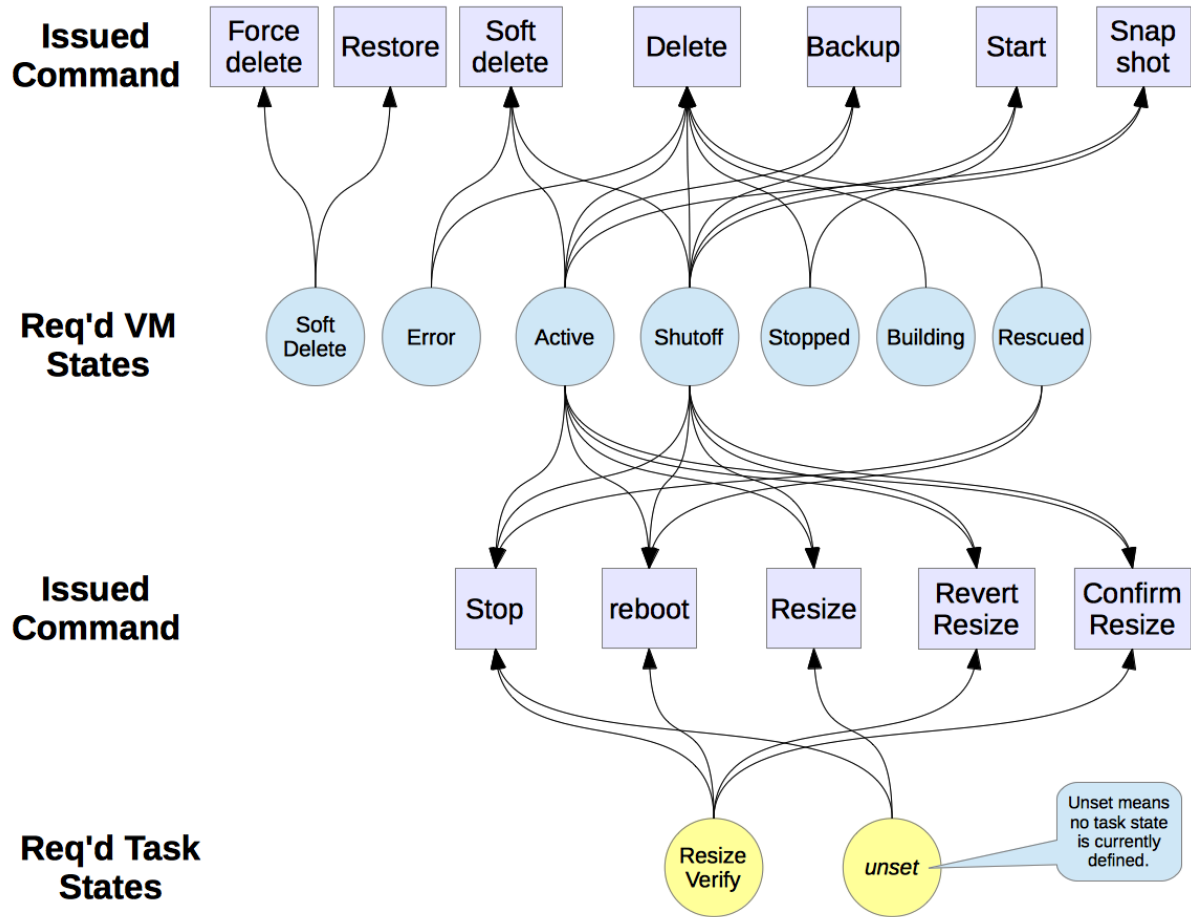
3.3.4 Virtual Machine States and Transitions

Preconditions for commands

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

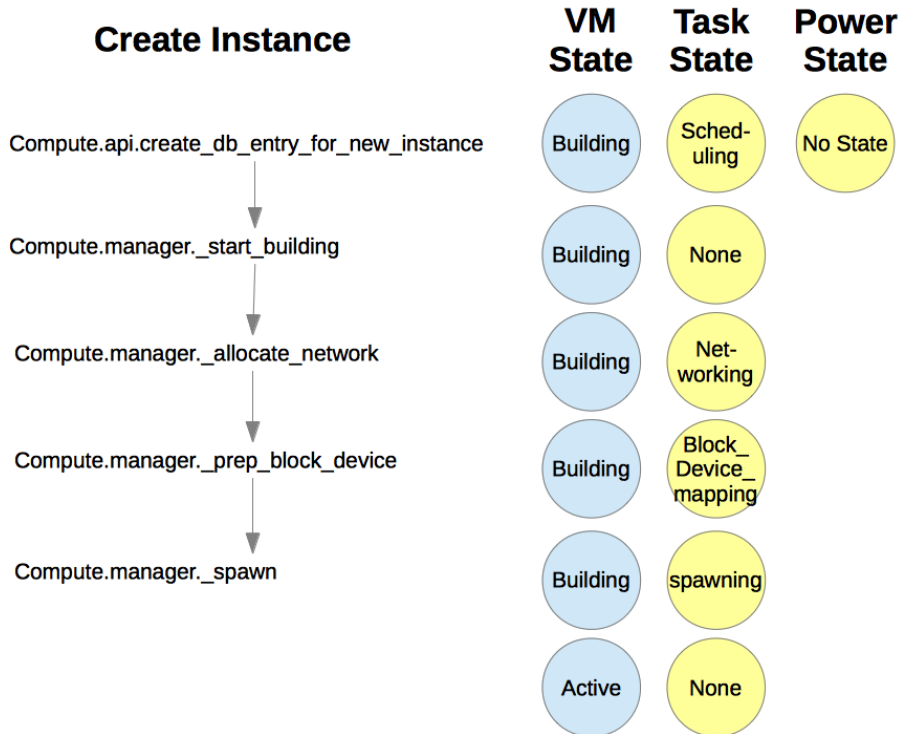






Create instance states

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.

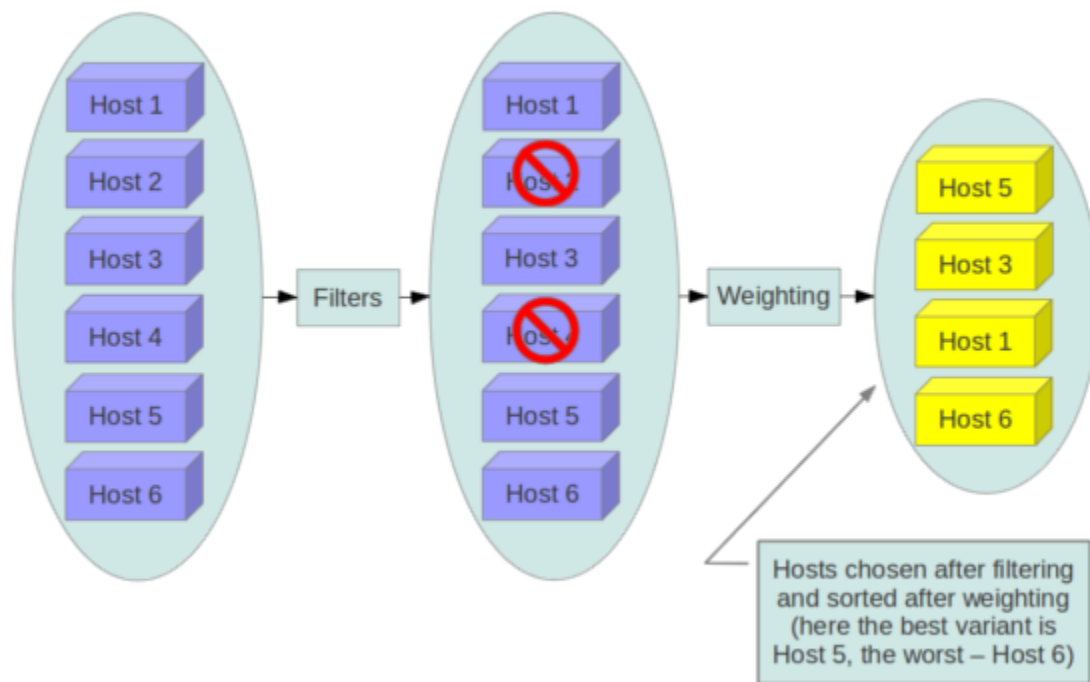
The `_()`, `_LC()`, `_LE()`, `_LW()` and `_LI()` functions can be imported with:

```
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 meta-data. 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`.

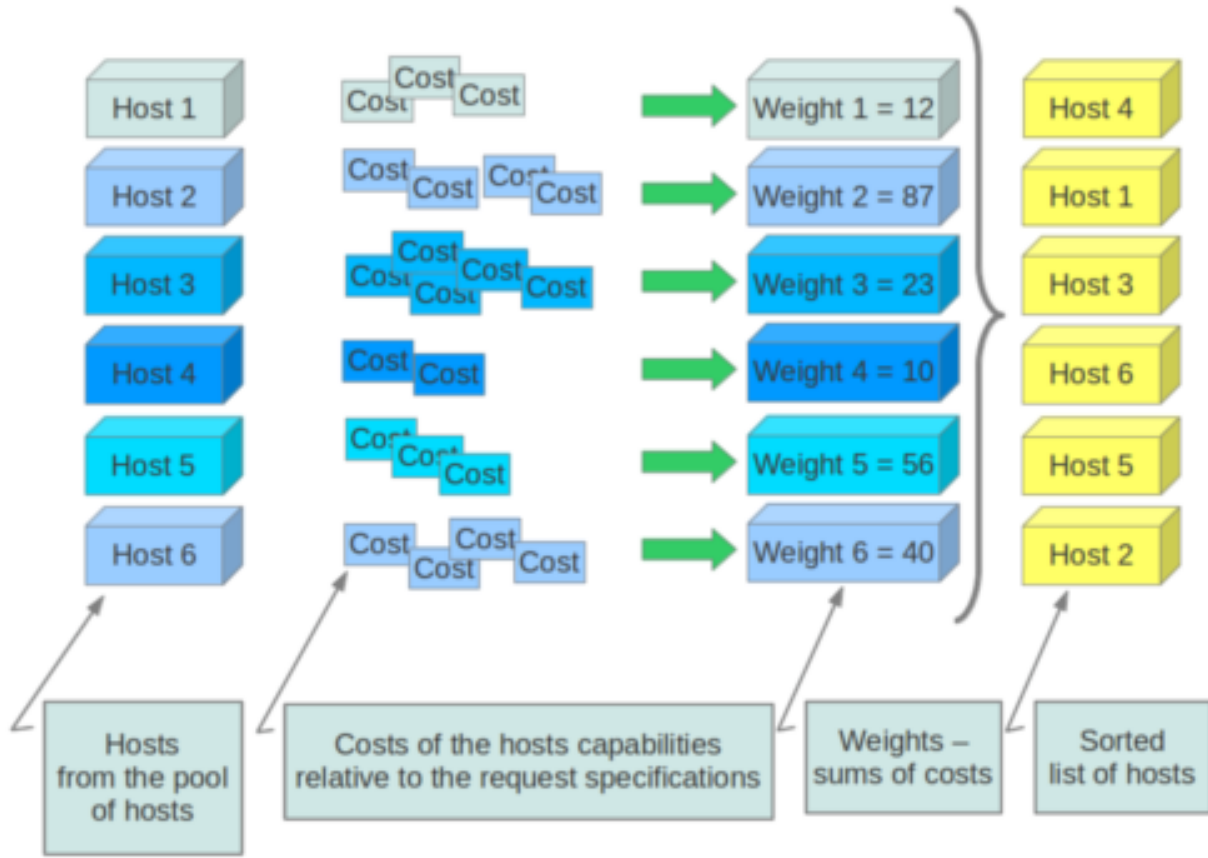
The Filter Scheduler weighs hosts based on the config option `scheduler_weight_classes`, this defaults to `nova.scheduler.weights.all_weighers`, which selects the following weighers:

- `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.

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

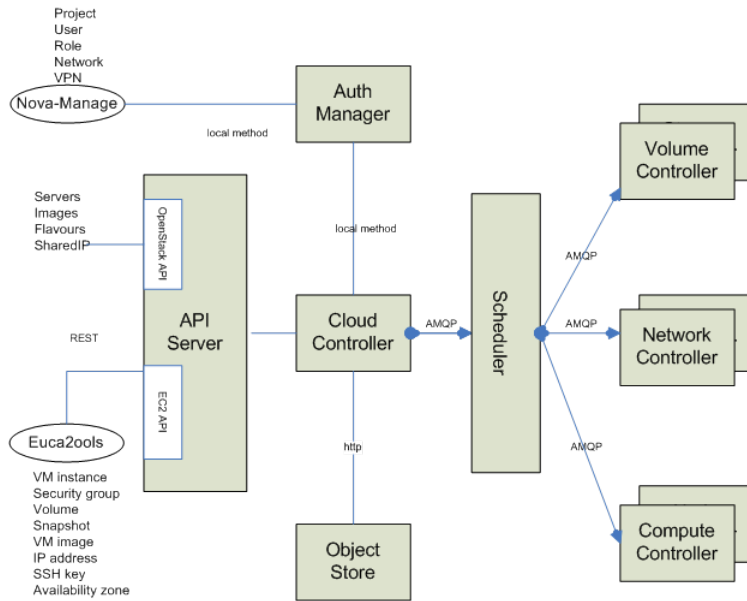
3.3.7 AMQP and Nova

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

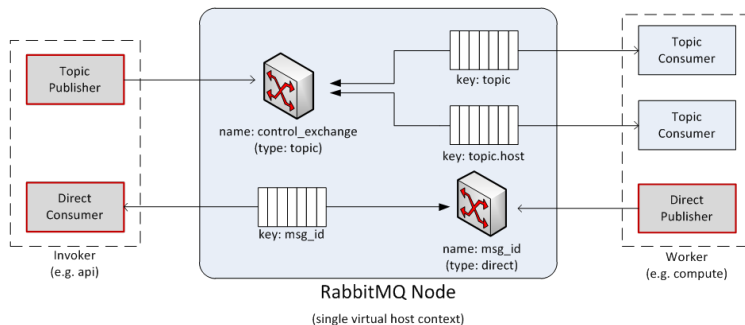
The figure below shows the internals of a message broker node (referred to as a RabbitMQ node in the diagrams) when a single instance is deployed and shared in an OpenStack cloud. Every Nova component connects to the message broker and, depending on its personality (for example a compute node or a network 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).

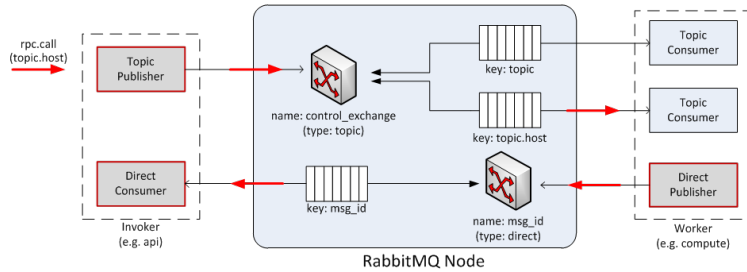
- **Topic Consumer:** a Topic Consumer comes to life as soon as a Worker is instantiated and exists throughout its life-cycle; this object is used to receive messages from the queue and it invokes the appropriate action as defined by the Worker role. A Topic Consumer connects to the same topic-based 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:

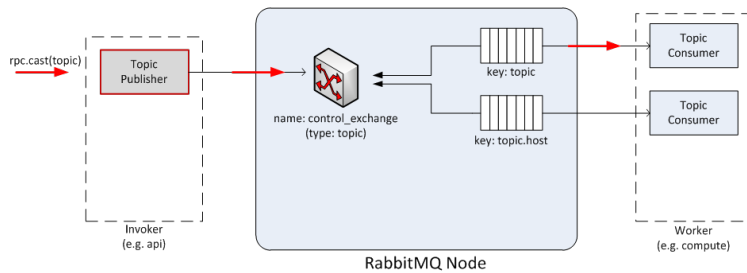
1. a Topic Publisher is instantiated to send the message request to the queuing system; immediately before the publishing operation, a Direct Consumer is instantiated to wait for the response message.
2. once the message is dispatched by the exchange, it is fetched by the Topic Consumer dictated by the 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

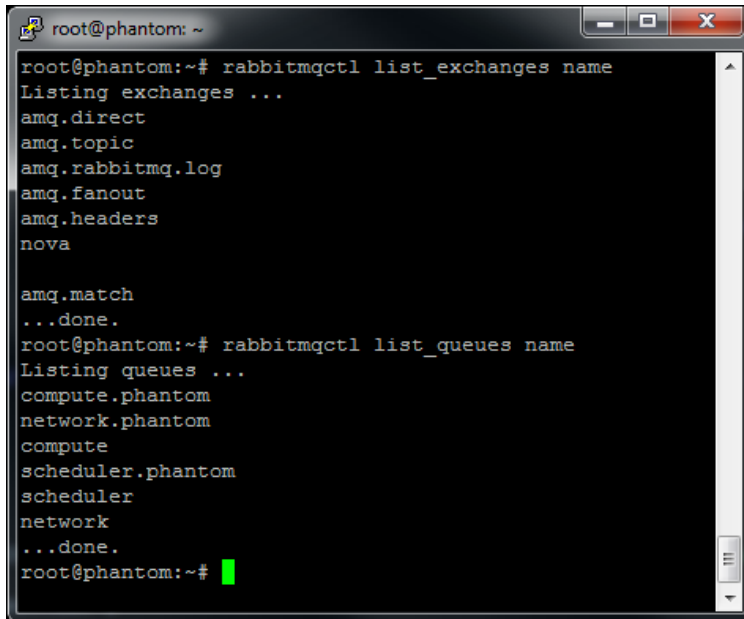
At any given time the load of a message broker node running either Qpid or RabbitMQ is function of the following parameters:

- **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



```

root@phantom: ~
root@phantom:~# rabbitmqctl list_exchanges name
Listing exchanges ...
amq.direct
amq.topic
amq.rabbitmq.log
amq.fanout
amq.headers
nova

amq.match
...done.
root@phantom:~# rabbitmqctl list_queues name
Listing queues ...
compute.phantom
network.phantom
compute
scheduler.phantom
scheduler
network
...done.
root@phantom:~#

```

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/flatten-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 than 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 $\geq 1.2.9$ and QEMU $\geq 2.1.2$

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

# rmmmod 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

# rmmmod 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/F
# 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 ml.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 scheduler 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, ComputeCapa
```

```
[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>
  <vcupin vcpu='0' cpuset='0-3' />

```

```
<vcupin vcpu='1' cpuset='0-3'/>
<vcupin vcpu='2' cpuset='0-3'/>
<vcupin 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 environment.

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-xvpngproxy`

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-xvpngproxy [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.

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*