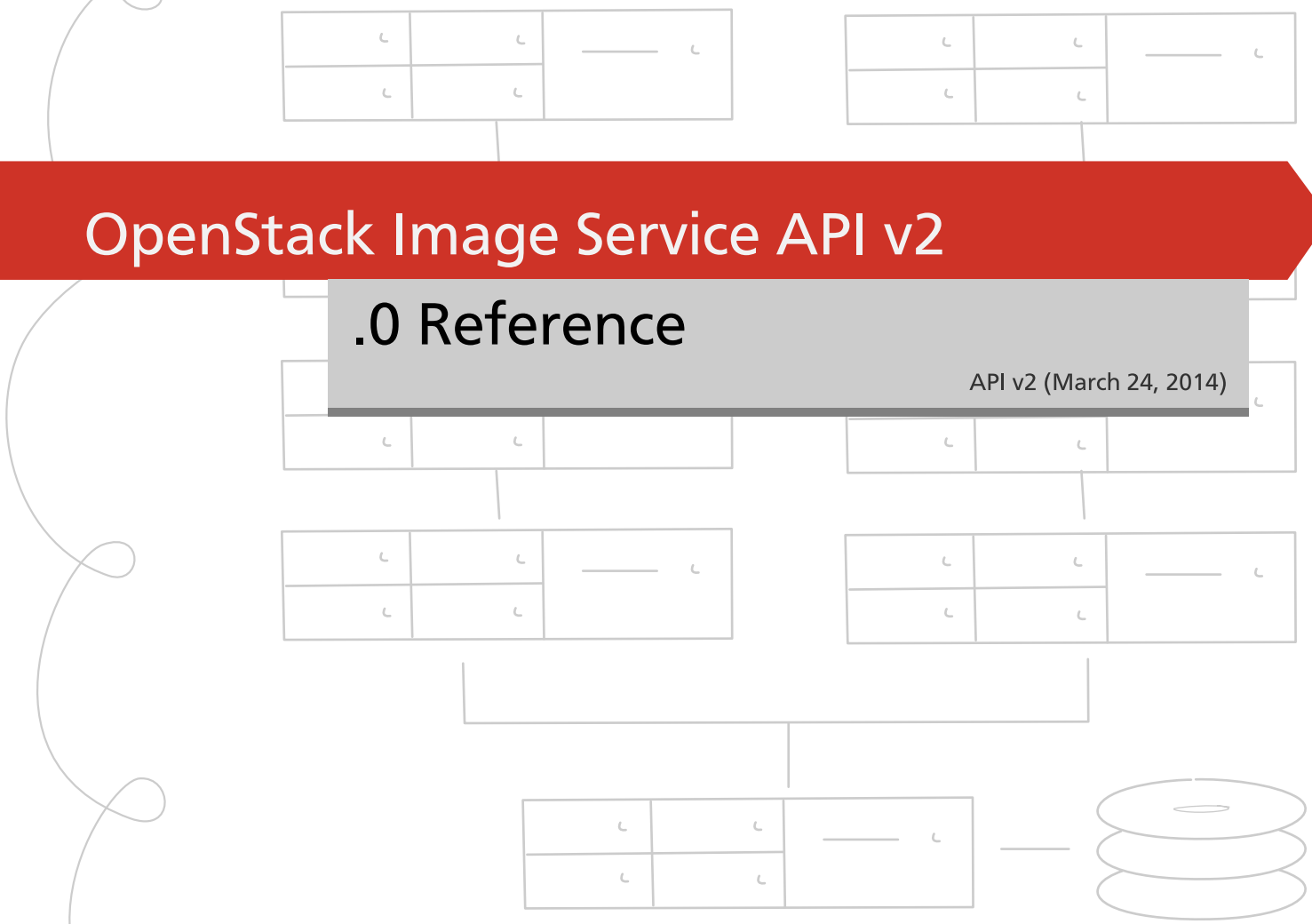
# OpenStack Image Service API v2

## .0 Reference

API v2 (March 24, 2014)



BUILT FOR

openstack™

CLOUD SOFTWARE

openstack™

docs.openstack.org

# OpenStack Image Service API v2.0 Reference

API v2 (2014-03-24)


This document is for software developers who develop applications by using the OpenStack™ Image Service Application Programming Interface (API) v2.

# Table of Contents

# 1. OpenStack Image Service API v2 Reference

## Table of Contents



# OpenStack Image Service API v2 Reference

The Image Service API v2 provides methods for storing and retrieving disk and server images.

# General API Information

## Versioning

**Two-part versioning scheme**

The Image Service API v2 follows the lead of the v1 API and use a major version and a minor version. For example, 'v2.3' would break down to major version 2 and minor version 3.

**Backwards-compatibility**

The interface will not be reduced with subsequent minor version releases, it will only be expanded. For example, everything in v2.1 will be available in v2.2.

**Property Protections**

Version 2.2 of the Images API acknowledges the ability of a cloud provider to employ *property protections*, an optional feature whereby CRUD protections may be applied to image properties. Thus, in particular deployments, non-admin users may not be able to view, update, or delete some image properties. Additionally, non-admin users may be forced to follow a particular naming convention when creating custom image properties. It is left to the cloud provider to communicate policies concerning property protections to users.

## HTTP Response Status Codes

The following HTTP status codes are all valid responses:

- 200 - generic successful response, expect a body

- 201 - entity created, expect a body and a Location header
- 204 - successful response without body
- 301 - redirection
- 400 - invalid request (syntax, value, etc)
- 401 - unauthenticated client
- 403 - authenticated client unable to perform action
- 409 - that action is impossible due to some (possibly permanent) circumstance
- 415 - unsupported media type

Responses that don't have a 200-level response code are not guaranteed to have a body. If a response does happen to return a body, it is not part of this spec and cannot be depended upon.

# Authentication and Authorization

This spec does not govern how one might authenticate or authorize clients of the v2 Images API. Implementors are free to decide how to identify clients and what authorization rules to apply.

Note that the HTTP status codes 401 and 403 are included in this specification as valid response codes.

# Request/Response Content Format

The v2 Images API primarily accepts and serves JSON-encoded data. In certain cases it also accepts and serves binary image data. Most requests that send JSON-encoded data must have the proper media type in their Content-Type header: 'application/json'. HTTP PATCH requests must use the patch media type defined for the entity they intend to modify. Requests that upload image data should use the media type 'application/octet-stream'.

Each call only responds in one format, so clients should not worry about sending an Accept header. It will be ignored. Assume a response will be formatted as 'application/json' unless otherwise stated in this spec.

# Image Entities

An image entity is represented by a JSON-encoded data structure and its raw binary data.

An image entity has an identifier (ID) that is guaranteed to be unique within the endpoint to which it belongs. The ID is used as a token in request URIs to interact with that specific image.

An image is always guaranteed to have the following attributes: id, status, visibility, protected, tags, created_at, file and self. The other attributes defined in the `image` schema below are guaranteed to be defined, but will only be returned with an image entity if they have been explicitly set.

A client may set arbitrarily-named attributes on their images if the `image` json-schema allows it. These user-defined attributes will appear like any other image attributes. See documentation of the additionalProperties json-schema attribute.

## JSON Schemas

The necessary json-schema documents will be provided at predictable URIs. A consumer should be able to validate server responses and client requests based on the published schemas. The schemas contained in this document are only examples and should not be used to validate your requests. A client should **always** fetch schemas from the server.

# Metadata API

The following calls allow you to create, modify, and delete image metadata records. For binary image data, see Binary Data API.

## Get Images Schema

**GET /v2/schemas/images**

Request body ignored.

Response body will contain a json-schema document representing an `images` entity (a container of `image` entities). For example:

```
{
    "name": "images",
    "properties": {
        "images": {
            "items": {
                "type": "array",
                "name": "image"
                "properties": {
                    "id": {"type": "string"},
                    "name": {"type": "string"},
                    "visibility": {"enum": ["public", "private"]},
                    "status": {"type": "string"},
                    "protected": {"type": "boolean"},
                    "tags": {
                        "type": "array",
                        "items": {"type": "string"}
                    },
                    "checksum": {"type": "string"},
                    "size": {"type": "integer"},
                    "created_at": {"type": "string"},
                    "updated_at": {"type": "string"},
                    "file": {"type": "string"},
                    "self": {"type": "string"},
                    "schema": {"type": "string"}
                },
                "additionalProperties": {"type": "string"},
                "links": [
                    {"href": "{self}", "rel": "self"},
                    {"href": "{file}", "rel": "enclosure"},
```

```
                    {"href": "{schema}", "rel": "describedby"}
                ]
            },
        },
        "schema": {"type": "string"},
        "next": {"type": "string"},
        "first": {"type": "string"}
    },
    "links": [
        {"href": "{first}", "rel": "first"},
        {"href": "{next}", "rel": "next"},
        {"href": "{schema}", "rel": "describedby"}
    ]
}
```

# Get Image Schema

**GET /v2/schemas/image**

Request body ignored.

Response body will contain a json-schema document representing an `image`. For example:

```
{
    "name": "image",
    "properties": {
        "id": {"type": "string"},
        "name": {"type": "string"},
        "visibility": {"enum": ["public", "private"]},
        "status": {"type": "string"},
        "protected": {"type": "boolean"},
        "tags": {
            "type": "array",
            "items": {"type": "string"}
        },
        "checksum": {"type": "string"},
        "size": {"type": "integer"},
        "created_at": {"type": "string"},
        "updated_at": {"type": "string"},
        "file": {"type": "string"},
        "self": {"type": "string"},
        "schema": {"type": "string"}
    },
    "additionalProperties": {"type": "string"},
    "links": [
        {"href": "{self}", "rel": "self"},
        {"href": "{file}", "rel": "enclosure"},
        {"href": "{schema}", "rel": "describedby"}
    ]
}
```

# Create an Image

**POST /v2/images**

Request body must be JSON-encoded and conform to the `image` JSON schema. For example:

```
{
    "id": "e7db3b45-8db7-47ad-8109-3fb55c2c24fd",
    "name": "Ubuntu 12.10",
    "tags": ["ubuntu", "quantal"]
}
```

Successful HTTP response will be 201 Created with a Location header containing the newly-created URI for the image. Response body will represent the created `image` entity. For example:

```
{
    "id": "e7db3b45-8db7-47ad-8109-3fb55c2c24fd",
    "name": "Ubuntu 12.10",
    "status": "queued",
    "visibility": "public",
    "tags": ["ubuntu", "quantal"],
    "created_at": "2012-08-11T17:15:52Z",
    "updated_at": "2012-08-11T17:15:52Z",
    "self": "/v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd",
    "file": "/v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd/file",
    "schema": "/v2/schemas/image"
}
```

# Update an Image

**PATCH /v2/images/<IMAGE_ID>**

Request body must conform to the 'application/openstack-images-v2.1-json-patch' media type, documented in Appendix B. Using **PATCH /v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd** as an example:

```
[
    {"op": "replace", "path": "/name", "value": "Fedora 17"},
    {"op": "replace", "path": "/tags", "value": ["fedora", "beefy"]}
]
```

Response body will represent the updated `image` entity. For example:

```
{
    "id": "e7db3b45-8db7-47ad-8109-3fb55c2c24fd",
    "name": "Fedora 17",
    "status": "queued",
    "visibility": "public",
    "tags": ["fedora", "beefy"],
    "created_at": "2012-08-11T17:15:52Z",
    "updated_at": "2012-08-11T17:15:52Z",
    "self": "/v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd",
    "file": "/v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd/file",
    "schema": "/v2/schemas/image"
}
```

The PATCH method can also be used to add or remove image properties. To add a custom user-defined property such as "login-user" to an image, use the following example request.

```
[
    {"op": "add", "path": "/login-user", "value": "kvothe"}
]
```

Similarly, to remove a property such as "login-user" from an image, use the following example request.

```
[
    {"op": "remove", "path": "/login-user"}
]
```

See Appendix B for more details about the 'application/openstack-images-v2.1-json-patch' media type.

**Property Protections**

Version 2.2 of the Images API acknowledges the ability of a cloud provider to employ *property protections*. Thus, there may be image properties that may not be updated or deleted by non-admin users.

# Add an Image Tag

**PUT /v2/images/<IMAGE_ID>/tags/<TAG>**

The the tag you want to add should be encoded into the request URI. For example, to tag image e7db3b45-8db7-47ad-8109-3fb55c2c24fd with 'miracle', you would **PUT /v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd/tags/miracle**. The request body is ignored.

An image tag may be up to 255 characters in length. See the 'image' json-schema to determine which characters are allowed.

An image can only be tagged once with a specific string. Multiple attempts to tag an image with the same string will result in a single instance of that string being added to the image's tags list.

An HTTP status of 204 will be returned.

# Delete an Image Tag

**DELETE /v2/images/<IMAGE_ID>/tags/<TAG>**

The tag you want to delete should be encoded into the request URI. For example, to remove the tag 'miracle' from image e7db3b45-8db7-47ad-8109-3fb55c2c24fd, you would **DELETE /v2/images/e7db3b45-8db7-47ad-8109-3fb55c2c24fd/tags/miracle**. The request body is ignored.

An HTTP status of 204 will be returned. Subsequent attempts to delete the tag will result in a 404.

# List All Images

**GET /v2/images**

Request body ignored.

Response body will be a list of images available to the client. For example:


```
{
    "images": [
        {
            "id": "da3b75d9-3f4a-40e7-8a2c-bfab23927dea",
            "name": "cirros-0.3.0-x86_64-uec-ramdisk",
            "status": "active",
            "visibility": "public",
            "size": 2254249,
            "checksum": "2cec138d7dae2aa59038ef8c9aec2390",
            "tags": ["ping", "pong"],
            "created_at": "2012-08-10T19:23:50Z",
            "updated_at": "2012-08-10T19:23:50Z",
            "self": "/v2/images/da3b75d9-3f4a-40e7-8a2c-bfab23927dea",
            "file": "/v2/images/da3b75d9-3f4a-40e7-8a2c-bfab23927dea/file",
            "schema": "/v2/schemas/image"
        },
        {
            "id": "0d5bcbc7-b066-4217-83f4-7111a60a399a",
            "name": "cirros-0.3.0-x86_64-uec",
            "status": "active",
            "visibility": "public",
            "size": 25165824,
            "checksum": "2f81976cae15c16ef0010c51e3a6c163",
            "tags": [],
            "created_at": "2012-08-10T19:23:50Z",
            "updated_at": "2012-08-10T19:23:50Z",
            "self": "/v2/images/0d5bcbc7-b066-4217-83f4-7111a60a399a",
            "file": "/v2/images/0d5bcbc7-b066-4217-83f4-7111a60a399a/file",
            "schema": "/v2/schemas/image"
        },
        {
            "id": "e6421c88-b1ed-4407-8824-b57298249091",
            "name": "cirros-0.3.0-x86_64-uec-kernel",
            "status": "active",
            "visibility": "public",
            "size": 4731440,
            "checksum": "cfb203e7267a28e435dbcb05af5910a9",
            "tags": [],
            "created_at": "2012-08-10T19:23:49Z",
            "updated_at": "2012-08-10T19:23:49Z",
            "self": "/v2/images/e6421c88-b1ed-4407-8824-b57298249091",
            "file": "/v2/images/e6421c88-b1ed-4407-8824-b57298249091/file",
            "schema": "/v2/schemas/image"
        }
    ],
    "first": "/v2/images?limit=3",
    "next": "/v2/images?limit=3&marker=e6421c88-b1ed-4407-8824-b57298249091",
    "schema": "/v2/schemas/images"
}
```


**Pagination**

This call is designed to return a subset of the larger collection of images while providing a link that can be used to retrieve the next. You should always check for the presence of a 'next' link and use it as the URI in a subsequent HTTP GET request. You should follow this

pattern until there a 'next' link is no longer provided. The next link will preserve any query parameters you send in your initial request. The 'first' link can be used to jump back to the first page of the collection.

If you prefer to paginate through images manually, the API provides two query parameters: 'limit' and 'marker'. The limit parameter is used to request a specific page size. Expect a response to a limited request to return between zero and *limit* items. The marker parameter is used to indicate the id of the last-seen image. The typical pattern of limit and marker is to make an initial limited request then to use the id of the last image from the response as the marker parameter in a subsequent limited request.

**Filtering**

The list operation accepts several types of query parameters intended to filter the results of the returned collection.

A client can provide direct comparison filters using *most* image attributes (i.e. name=Ubuntu, visibility=public, etc). A client cannot filter on tags or anything defined as a 'link' in the json-schema (i.e. self, file, schema).

The 'size_min' and 'size_max' query parameters can be used to do greater-than and less-than filtering of images based on their 'size' attribute ('size' is measured in bytes and refers to the size of an image when stored on disk). For example, sending a size_min filter of 1048576 and size_max of 4194304 would filter the container to include only images that are between one and four megabytes in size.

**Sorting**

The results of this operation can be ordered using the 'sort_key' and 'sort_dir' parameters. The API uses the natural sorting of whatever image attribute is provided as the 'sort_key'. All image attributes can be used as the sort_key (except tags and link attributes).
The sort_dir parameter indicates in which direction to sort. Acceptable values are 'asc' (ascending) and 'desc' (descending). Defaults values for sort_key and sort_dir are 'created_at' and 'desc'.

**Property Protections**

Version 2.2 of the Images API acknowledges the ability of a cloud provider to employ *property protections*. Thus, there may be image properties that will not appear in the list images response for non-admin users.

# Get an Image

**GET /v2/images/<IMAGE_ID>**

Request body ignored.

Response body will be a single image entity. Using **GET /v2/image/da3b75d9-3f4a-40e7-8a2c-bfab23927dea** as an example:

```
{
    "id": "da3b75d9-3f4a-40e7-8a2c-bfab23927dea",
    "name": "cirros-0.3.0-x86_64-uec-ramdisk",
```

```
    "status": "active",
    "visibility": "public",
    "size": 2254249,
    "checksum": "2cec138d7dae2aa59038ef8c9aec2390",
    "tags": ["ping", "pong"],
    "created_at": "2012-08-10T19:23:50Z",
    "updated_at": "2012-08-10T19:23:50Z",
    "self": "/v2/images/da3b75d9-3f4a-40e7-8a2c-bfab23927dea",
    "file": "/v2/images/da3b75d9-3f4a-40e7-8a2c-bfab23927dea/file",
    "schema": "/v2/schemas/image"
}
```

**Property Protections**

Version 2.2 of the Images API acknowledges the ability of a cloud provider to employ *property protections*. Thus, there may be some image properties that will not appear in the image detail response for non-admin users.

# Delete an Image

**DELETE /v2/images/<IMAGE_ID>**

Encode the ID of the image into the request URI. Request body is ignored.

Images with the 'protected' attribute set to true (boolean) cannot be deleted and the response will have an HTTP 403 status code. You must first set the 'protected' attribute to false (boolean) and then perform the delete.

The response will be empty with an HTTP 204 status code.

# Image Sharing

The OpenStack Image Service API v2 allows users to share images with each other.

Let the "producer" be a tenant who owns image 71c675ab-d94f-49cd-a114-e12490b328d9, and let the "consumer" be a tenant who would like to boot an instance from that image.

The producer can share the image with the consumer by making the consumer a **member** of that image.

To prevent spamming, the consumer must **accept** the image before it will be included in the consumer's image list.

The consumer can still boot from the image, however, if the consumer knows the image ID.

In summary:

- The image producer may add or remove image members, but may not modify the member status of an image member.
- An image consumer may change his or her member status, but may not add or remove him or herself as an image member.
- A consumer may boot an instance from a shared image regardless of whether he/she has "accepted" the image.

## Producer-Consumer Communication

No provision is made in this API for producer-consumer communication. All such communication must be done independently of the API.

An example workflow is:

1. The producer posts the availability of specific images on a public website.
2. A potential consumer provides the producer with his/her tenant ID and email address.
3. The producer uses the Images v2 API to share the image with the consumer.
4. The producer notifies the consumer via email that the image has been shared and what its UUID is.
5. If the consumer wishes the image to appear in his/her image list, the Images v2 API is used to change the image status to `accepted`.
6. If the consumer subsequently wishes to hide the image, the Images v2 API may be used to change the member status to `rejected`. If the consumer wishes to hide the image, but is open to the possibility of being reminded by the producer that the image is available, the Images v2 API may be used to change the member status to `pending`.

Note that as far as this API is concerned, the member status has only two effects:

- If the member status is *not* `accepted`, the image will not appear in the consumer's default image list.
- The consumer's image list may be filtered by status to see shared images in the various member statuses. For example, the consumer can discover images that have been shared with him or her by filtering on `visibility=shared&member_status=pending`.

## Image Sharing Schemas

JSON schema documents are provided at the URIs listed below.

Recall that the schemas contained in this document are only examples and should not be used to validate your requests.

### Get Image Member Schema

**GET /v2/schemas/member**

Request body ignored.

Response body contains a json-schema document representing an image `member` entity.

The response from the API should be considered authoritative. The schema is reproduced here solely for your convenience:

```
{
    "name": "member",
    "properties": {
        "created_at": {
```

```
            "description": "Date and time of image member creation",
            "type": "string"
        },
        "image_id": {
            "description": "An identifier for the image",
            "pattern": "^([0-9a-fA-F]){8}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-
([0-9a-fA-F]){4}-([0-9a-fA-F]){12}$",
            "type": "string"
        },
        "member_id": {
            "description": "An identifier for the image member (tenantId)",
            "type": "string"
        },
        "status": {
            "description": "The status of this image member",
            "enum": [
                "pending",
                "accepted",
                "rejected"
            ],
            "type": "string"
        },
        "updated_at": {
            "description": "Date and time of last modification of image
 member",
            "type": "string"
        },
        "schema": {
            "type": "string"
        }
    }
}
```

## Get Image Members Schema

**GET /v2/schemas/members**

Request body ignored.

Response body contains a json-schema document representing an image `members` entity (a container of `member` entities).

The response from the API should be considered authoritative. The schema is reproduced here solely for your convenience:

```
{
    "name": "members",
    "properties": {
        "members": {
            "items": {
                "name": "member",
                "properties": {
                    "created_at": {
                        "description": "Date and time of image member
 creation",
                        "type": "string"
                    },
                    "image_id": {
                        "description": "An identifier for the image",
```

```
                        "pattern": "^([0-9a-fA-F]){8}-([0-9a-fA-F]){4}-([0-9a-
fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){12}$",
                        "type": "string"
                    },
                    "member_id": {
                        "description": "An identifier for the image member
 (tenantId)",
                        "type": "string"
                    },
                    "status": {
                        "description": "The status of this image member",
                        "enum": [
                            "pending",
                            "accepted",
                            "rejected"
                        ],
                        "type": "string"
                    },
                    "updated_at": {
                        "description": "Date and time of last modification of
 image member",
                        "type": "string"
                    },
                    "schema": {
                        "type": "string"
                    }
                }
            },
            "type": "array"
        },
        "schema": {
            "type": "string"
        }
    },
    "links": [
        {
            "href": "{schema}",
            "rel": "describedby"
        }
    ]
}
```

## Image Producer Calls

The following calls are germane to a user who wishes to act as a producer of shared images.

### Create an Image Member

**POST /v2/images/<IMAGE_ID>/members**

The request body must be JSON in the following format:

```
{
    "member": "<MEMBER_ID>"
}
```

where the MEMBER_ID is the ID of the tenant with whom the image is to be shared.

The member status of a newly created image member is `pending`.

If the user making the call is not the image owner, the response is HTTP status code 404.

The response conforms to the JSON schema available at **/v2/schemas/member**, for example,

```
{
    "created_at": "2013-09-19T20:36:53Z",
    "image_id": "71c675ab-d94f-49cd-a114-e12490b328d9",
    "member_id": "8989447062e04a818baf9e073fd04fa7",
    "schema": "/v2/schemas/member",
    "status": "pending",
    "updated_at": "2013-09-19T20:36:53Z"
}
```

### Delete an Image Member

**DELETE /v2/images/<IMAGE_ID>/members/<MEMBER_ID>**

A successful response is 204 (No Content).

The call returns HTTP status code 404 if MEMBER_ID is not an image member of the specified image.

The call returns HTTP status code 404 if the user making the call is not the image owner.

## Image Consumer Calls

The following calls pertain to a user who wishes to act as a consumer of shared images.

### Update an Image Member

**PUT /v2/images/<IMAGE_ID>/members/<MEMBER_ID>**

The body of the request is a JSON object specifying the member status to which the image member should be updated:

```
{
    "status": "<STATUS_VALUE>"
}
```

where STATUS_VALUE is one of { `pending`, `accepted`, or `rejected` }.

The response conforms to the JSON schema available at **/v2/schemas/member**, for example,

```
{
    "created_at": "2013-09-20T19:22:19Z",
    "image_id": "a96be11e-8536-4910-92cb-de50aa19dfe6",
    "member_id": "8989447062e04a818baf9e073fd04fa7",
    "schema": "/v2/schemas/member",
    "status": "accepted",
    "updated_at": "2013-09-20T20:15:31Z"
}
```

If the call is made by the image owner, the response is HTTP status code 403 (Forbidden).

If the call is made by a user who is not the image owner and whose tenant ID does not match the MEMBER_ID, the response is HTTP status code 404.

### Image Member Status Values

There are three image member status values:

- `pending`: When a member is created, its status is set to `pending`. The image is not visible in the member's image-list, but the member can still boot instances from the image.
- `accepted`: When a member's status is `accepted`, the image is visible in the member's image-list. The member can boot instances from the image.
- `rejected`: When a member's status is `rejected`, the member has decided that he or she does not wish to see the image. The image is not visible in the member's image-list, but the member can still boot instances from the image.

## Calls for Both Producers and Consumers

These calls are applicable to users acting either as producers or consumers of shared images.

### Show Image Member

**GET /v2/images/<IMAGE_ID>/members/<MEMBER_ID>**

The response conforms to the JSON schema available at **/v2/schemas/member**, for example,

```
{
    "created_at": "2014-02-20T04:15:17Z",
    "image_id": "634985e5-0f2e-488e-bd7c-928d9a8ea82a",
    "member_id": "46a12bfd09c8459483c03e1b0d71bda8",
    "schema": "/v2/schemas/member",
    "status": "pending",
    "updated_at": "2014-02-20T04:15:17Z"
}
```

The image owner (the producer) may make this call successfully for each image member. An image member (a consumer) may make this call successfully only when MEMBER_ID matches that consumer's tenant ID. For any other MEMBER_ID, the consumer receives a 404 response.

### List Image Members

**GET /v2/images/<IMAGE_ID>/members**

The response conforms to the JSON schema available at **/v2/schemas/members**, for example,

```
{
    "members": [
        {
            "created_at": "2013-09-20T19:16:53Z",
            "image_id": "a96be11e-8536-4910-92cb-de50aa19dfe6",
            "member_id": "818baf9e073fd04fa78989447062e04a",
            "schema": "/v2/schemas/member",
            "status": "pending",
            "updated_at": "2013-09-20T19:16:53Z"
        },
        {
            "created_at": "2013-09-20T19:22:19Z",
            "image_id": "a96be11e-8536-4910-92cb-de50aa19dfe6",
            "member_id": "8989447062e04a818baf9e073fd04fa7",
            "schema": "/v2/schemas/member",
            "status": "pending",
            "updated_at": "2013-09-20T19:22:19Z"
        }
    ],
    "schema": "/v2/schemas/members"
}
```


If the call is made by a user with whom the image has been shared, the member-list will contain *only* the information for that user. For example, if the call is made by tenant 8989447062e04a818baf9e073fd04fa7, the response is:


```
{
    "members": [
        {
            "created_at": "2013-09-20T19:22:19Z",
            "image_id": "a96be11e-8536-4910-92cb-de50aa19dfe6",
            "member_id": "8989447062e04a818baf9e073fd04fa7",
            "schema": "/v2/schemas/member",
            "status": "pending",
            "updated_at": "2013-09-20T19:22:19Z"
        }
    ],
    "schema": "/v2/schemas/members"
}
```


If the call is made by a user with whom the image is *not* shared, the response is a 404.

## List Shared Images

Shared images are listed as part of the normal image list call. In this section we emphasize some useful filtering options.

- `visibility=shared`: show only images shared with me where my member status is 'accepted'
- `visibility=shared&member_status=accepted`: same as above
- `visibility=shared&member_status=pending`: show only images shared with me where my member status is 'pending'
- `visibility=shared&member_status=rejected`: show only images shared with me where my member status is 'rejected'

- `visibility=shared&member_status=all`: show all images shared with me regardless of my member status
- `owner=<OWNER_ID>`: show only images shared with me by the user whose tenant ID is OWNER_ID

# Binary Data API

The following API calls are used to upload and download raw image data. For image metadata, see Metadata API.

## Store Image File

**PUT /v2/images/<IMAGE_ID>/file**

NOTE: An image record must exist before a client can store binary image data with it.

Request Content-Type must be 'application/octet-stream'. Complete contents of request body will be stored and become accessible in its entirety by issuing a GET request to the same URI.

Response status will be 204.

## Get Image File

**GET /v2/images/<IMAGE_ID>/file**

Request body ignored.

Response body will be the raw binary data that represents the actual virtual disk. The Content-Type header will be 'application/octet-stream'.

The Content-MD5 header will contain an MD5 checksum of the image data. Clients are encouraged to verify the integrity of the image data they receive using this checksum.

If no image data has been stored, an HTTP status of 204 will be returned.

# Appendix A: cURL Examples

This section is intended to provide a series of commands a typical client of the API might use to create and modify an image.

These commands assume the implementation of the v2 Images API is using the OpenStack Identity Service for authentication and authorization. The X-Auth-Token header is used to communicate the authentication token provided by that separate identity service.

The strings `$OS_IMAGE_URL` and `$OS_AUTH_TOKEN` represent variables defined in the client's environment. `$OS_IMAGE_URL` is the full path to your image service endpoint, for example, `http://localhost:9292`. `$OS_AUTH_TOKEN`

represents an auth token generated by the OpenStack Identity Service, for example, `6583fb17c27b48b4b4a6033fe9cc0fe0`.

# Create an Image

```
% curl -i -X POST -H "X-Auth-Token: $OS_AUTH_TOKEN" \
       -H "Content-Type: application/json" \
       -d '{"name": "Ubuntu 12.10", "tags": ["ubuntu", "12.10", "quantal"]}' \
       $OS_IMAGE_URL/v2/images
```

```
HTTP/1.1 201 Created
Content-Length: 451
Content-Type: application/json; charset=UTF-8
Location: http://localhost:9292/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad
Date: Tue, 14 Aug 2012 00:46:48 GMT

{
    "id": "7b97f37c-899d-44e8-aaa0-543edbc4eaad",
    "name": "Ubuntu 12.10",
    "status": "queued",
    "visibility": "private",
    "protected": false,
    "tags": ["ubuntu", "12.10", "quantal"],
    "created_at": "2012-08-14T00:46:48Z",
    "updated_at": "2012-08-14T00:46:48Z",
    "file": "/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad/file",
    "self": "/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad",
    "schema": "/v2/schemas/image"
}
```

# Update the Image

```
% curl -i -X PATCH -H "X-Auth-Token: $OS_AUTH_TOKEN" \
       -H "Content-Type: application/openstack-images-v2.1-json-patch"
       -d '[{"op": "add", "path": "/login-user", "value": "root"}]' \
       $OS_IMAGE_URL/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad
```

```
HTTP/1.1 200 OK
Content-Length: 477
Content-Type: application/json; charset=UTF-8
Date: Fri, 15 Nov 2013 00:46:50 GMT

{
    "id": "7b97f37c-899d-44e8-aaa0-543edbc4eaad",
    "name": "Ubuntu 12.10",
    "status": "queued",
    "visibility": "private",
    "protected": false,
    "tags": ["ubuntu", "12.10", "quantal"],
    "login_user": "root",
    "created_at": "2013-11-15T00:46:48Z",
    "updated_at": "2013-11-15T00:46:50Z",
    "file": "/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad/file",
    "self": "/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad",
    "schema": "/v2/schemas/image"
```

```
}
```

## Upload Binary Image Data

```
% curl -i -X PUT -H "X-Auth-Token: $OS_AUTH_TOKEN" \
       -H "Content-Type: application/octet-stream" \
       -d @/home/glance/ubuntu-12.10.qcow2 \
       $OS_IMAGE_URL/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad/file
```

```
HTTP/1.1 100 Continue

HTTP/1.1 201 Created
Content-Length: 0
Date: Tue, 14 Aug 2012 00:46:59 GMT
```

## Download Binary Image Data

```
% curl -i -X GET -H "X-Auth-Token: $OS_AUTH_TOKEN" \
       $OS_IMAGE_URL/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad/file
```

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Md5: 912ec803b2ce49e4a541068d495ab570
Transfer-Encoding: chunked
Date: Thu, 14 Aug 2012 00:47:10 GMT
```

## Delete Image

```
% curl -i -X DELETE -H "X-Auth-Token: $OS_AUTH_TOKEN" \
       $OS_IMAGE_URL/v2/images/7b97f37c-899d-44e8-aaa0-543edbc4eaad
```

```
HTTP/1.1 204 No Content
Content-Length: 0
Date: Tue, 14 Aug 2012 00:47:12 GMT
```

# Appendix B: HTTP PATCH media types

## Overview

The HTTP PATCH request must provide a media type for the server to determine how the patch should be applied to an image resource. An unsupported media type will result in an HTTP error response with the 415 status code. For image resources, two media types are supported:

- `application/openstack-images-v2.1-json-patch`
- `application/openstack-images-v2.0-json-patch`

The `application/openstack-images-v2.1-json-patch` media type is intended to provide a useful and compatible subset of the functionality defined in JavaScript Object

Notation (JSON) Patch RFC6902, which defines the `application/json-patch+json` media type.

The `application/openstack-images-v2.0-json-patch` media type is based on draft 4 of the standard. Its use is deprecated.

# Restricted JSON Pointers

The 'application/openstack-images-v2.1-json-patch' media type defined in this appendix adopts a restricted form of JSON-Pointers. A restricted JSON pointer is a Unicode string containing a sequence of exactly one reference token, prefixed by a '/' (%x2F) character.

If a reference token contains '~' (%x7E) or '/' (%x2F) characters, they must be encoded as '~0' and '~1' respectively.

Its ABNF syntax is:

```
restricted-json-pointer = "/" reference-token
reference-token = *( unescaped / escaped )
unescaped = %x00-2E / %x30-7D / %x7F-10FFFF
escaped = "~" ( "0" / "1" )
```

Restricted JSON Pointers are evaluated as ordinary JSON pointers per JSON-Pointer.

For example, given the `image` entity

```
{
    "id": "da3b75d9-3f4a-40e7-8a2c-bfab23927dea",
    "name": "cirros-0.3.0-x86_64-uec-ramdisk",
    "status": "active",
    "visibility": "public",
    "size": 2254249,
    "checksum": "2cec138d7dae2aa59038ef8c9aec2390",
    "~/.ssh/": "present",
    "tags": ["ping", "pong"],
    "created_at": "2012-08-10T19:23:50Z",
    "updated_at": "2012-08-10T19:23:50Z",
    "self": "/v2/images/da3b75d9-3f4a-40e7-8a2c-bfab23927dea",
    "file": "/v2/images/da3b75d9-3f4a-40e7-8a2c-bfab23927dea/file",
    "schema": "/v2/schemas/image"
}
```

the following restricted JSON pointers evaluate to the accompanying values:

```
"/name"        "cirros-0.3.0-x86_64-uec-ramdisk"
"/size"        2254249
"/tags"        ["ping", "pong"]
"/~0~1.ssh~1"  "present"
```

# Operations

The 'application/openstack-images-v2.1-json-patch' media type supports a subset of the operations defined in the 'application/json-patch+json' media type. The operation to perform is expressed as the value of the "op" member of the operation object.

- The operations supported are: "add", "remove", "replace".
- It is an error condition if an operation object contains no recognized operation member.

The location within the target image where the requested operation is to be performed is specified by using the "path" member of the operation object.

- The member value is a string containing a restricted JSON pointer value that references the location where the operation is to be performed within the target image.

Where appropriate (that is, for the "add" and "replace" operations), the operation object must contain a third data member, "value".

- The member value is the actual value to add (or to use in the replace operation) expressed in JSON notation. (For example, strings must be quoted, numeric values are unquoted.)

The payload for a PATCH request must be a *list* of json objects, each of which adheres to one of the formats described below.

- add

The "add" operation adds a new value at a specified location in the target image. The location must reference an image property to add to an existing image. The operation object contains a "value" member that specifies the value to be added.

Example:

```
{ "op": "add", "path": "/login-name", "value": "kvothe"}
```

- remove

The "remove" operation removes the specified image property in the target image. It is an error condition if no image property exists at the specified location.

Example:

```
{ "op": "remove", "path": "/login-name" }
```

- replace

The "replace" operation replaces the value of the specified image property in the target image with a new value. The operation object contains a "value" member that specifies the replacement value.

Example:

```
{ "op": "replace", "path": "/login-name", "value": "kote" }
```

This operation is functionally identical to expressing a "remove" operation for an image property, followed immediately by an "add" operation at the same location with the replacement value.

It is an error condition if the specified image property does not exist for the target image.