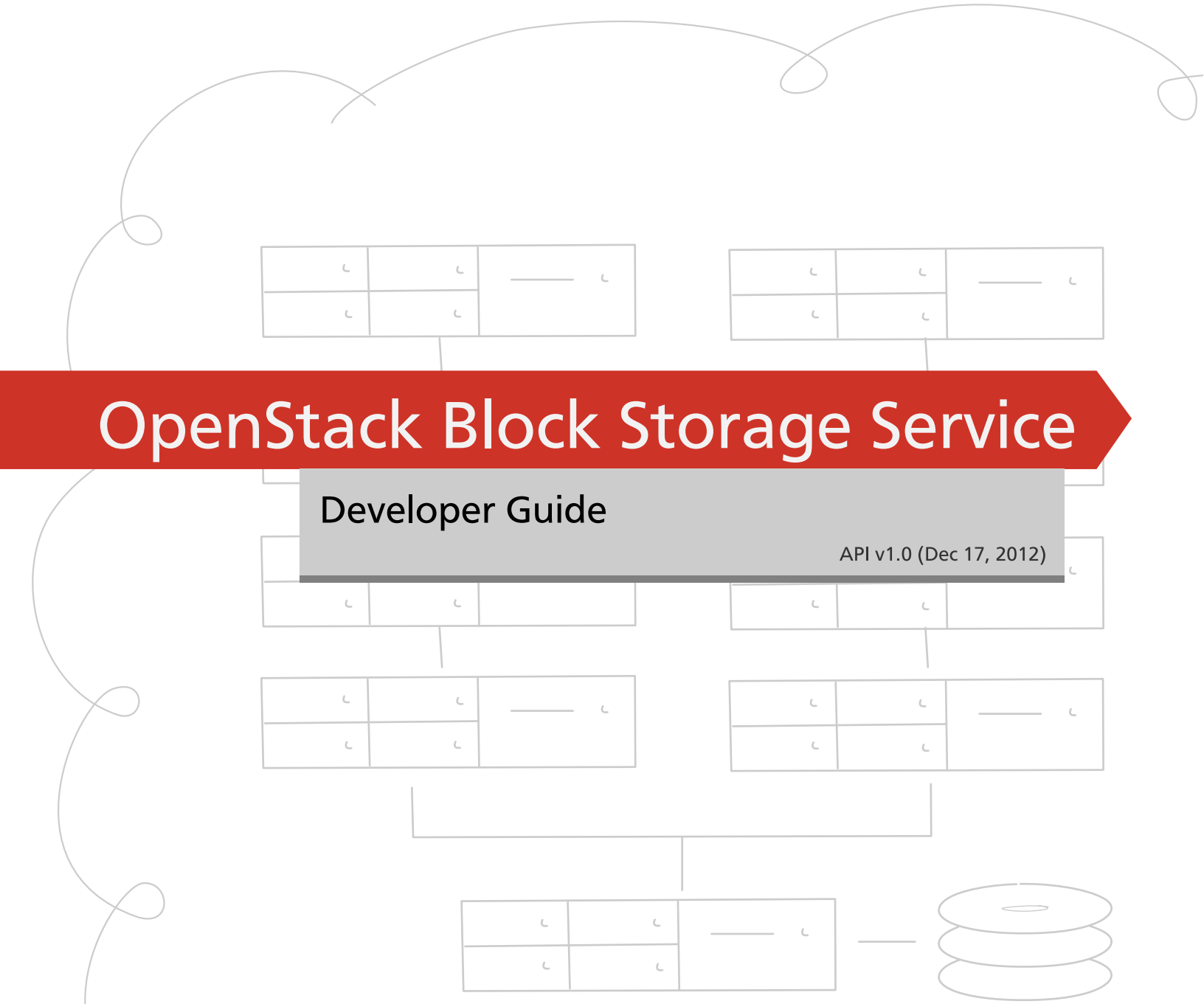

OpenStack Block Storage Service

Developer Guide

API v1.0 (Dec 17, 2012)

openstack™

docs.openstack.org

OpenStack Block Storage Service Developer Guide

API v1.0 (2012-12-17)

This document is intended for software developers interested in developing applications using the OpenStack Block Storage Service Application Programming Interface (API).

OpenStack API Reference

The OpenStack system has several key projects that are separate installations but can work together depending on your cloud needs: OpenStack Compute, OpenStack Object Storage, OpenStack Block Storage, OpenStack Identity Service, and OpenStack Image Service.

This page covers the basics for working with your OpenStack cloud through the Compute API and Image API after authorizing with the Identity Service API. You can then build a cloud by launching images and assigning metadata to instances, all through an API. Refer to the API Quick Start Guide for details about using the APIs referenced here, and go to docs.openstack.org/api for specifications of each API.

Overview

OpenStack Block Storage Service is a block-level storage solution that allows customers to mount drives or volumes to their OpenStack Cloud Servers™. The two primary use cases are (1) to allow customers to scale their storage independently from their compute resources, and (2) to allow customers to utilize high performance storage to serve database or I/O-intensive applications.

Interactions with Block Storage occur programmatically via the Block Storage API as described in this Developer Guide.

Highlights of OpenStack Block Storage Service include:

- Mount a drive to a Cloud Server to scale storage without paying for more compute capability.

Notes

- OpenStack Block Storage Service is an add-on feature to OpenStack Nova Compute.
- Block Storage is multi-tenant rather than dedicated.
- Block Storage allows you to create snapshots that you can save, list, and restore.

1. Intended Audience

This Guide is intended to assist software developers who want to develop applications using the Block Storage API. It assumes the reader has a general understanding of storage and is familiar with:

- ReSTful web services
- HTTP/1.1 conventions
- JSON and/or XML data serialization formats

2. Document Change History

This version of the Developer Guide replaces and obsoletes all previous versions. The most recent changes are described in the table below:

Revision Date	Summary of Changes
Dec 17, 2012	• Edits to initial version; .
Oct 17, 2012	• Initial release.

3. Additional Resources

You can download the most current versions of the API-related documents from docs.openstack.com/api/.

This API uses standard HTTP 1.1 response codes as documented at: www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

4. Concepts

To use the Block Storage API effectively, you should understand several key concepts:

4.1. Instance

An instance is a virtual machine that runs inside the cloud.

4.2. Instance Type

An instance type describes the compute, memory and storage capacity of Nova computing instances. In layman's terms, this is the size (in terms of vCPUs, RAM, and so forth) of the virtual server that you will be launching.

4.3. Snapshot

A snapshot is a point in time copy of the data contained in a volume.

4.4. Volume

A volume is a detachable block storage device. You can think of it as a USB hard drive. It can only be attached to one instance at a time.

4.5. Volume Type

The volume type is the type of a block storage volume. You may define whatever types work best for you, such as SATA, SCSCI, SSD, etc. These can be customized or defined by the OpenStack admin.

You may also define extra_specs associated with your volume types. For instance, you could have a VolumeType=SATA, with extra_specs (RPM=10000, RAID-Level=5) . Extra_specs are defined and customized by the admin.

1. General API Information

1.1. Authentication

Authentication tokens are typically valid for 24 hours. Applications should be designed to re-authenticate after receiving a 401 (Unauthorized) response from a service endpoint.

Important

If you are programmatically parsing an authentication response, please be aware that service names are stable for the life of the particular service and can be used as keys. You should also be aware that a user's service catalog can include multiple uniquely-named services which perform similar functions. For example, cloudServersOpenStack is the OpenStack version of compute whereas cloudServers is the legacy version of compute; the same user can have access to both services. In Auth 2.0, the service type attribute can be used as a key by which to recognize similar services; see the tip below.

Tip

Beginning with Auth 2.0, the service catalog includes a service type attribute to identify services that perform similar functions but have different names; for example, `type="compute"` identifies compute services such as cloudServers and cloudServersOpenStack. Some developers have found the service type attribute to be useful in parsing the service catalog. For Auth 2.0 (also known as the Cloud Identity Service), you can see the service type attribute in the "Service Catalog in Authentication Response" samples in the *Cloud Identity Client Developer Guide* at http://docs.openstack.com/auth/api/v2.0/auth-client-devguide/content/Sample_Request_Response-d1e64.html.

1.2. Service Access/Endpoints

The Block Storage Service is a regionalized service. The user of the service is therefore responsible for appropriate replication, caching, and overall maintenance of Block Storage data across regional boundaries to other Block Storage servers.

You can find the available service access/endpoints for Block Storage summarized in the table below.

Replace the sample account ID number, *1234*, with your actual account number returned as part of the authentication service response.

You will find the actual account number after the final '/' in the `publicURL` field returned by the authentication response.

1.3. Block Storage Service Versions

The Block Storage version defines the contract and build information for the API.

1.3.1. Contract Version

The contract version denotes the data model and behavior that the API supports. The requested contract version is included in all request URLs. Different contract versions of the API may be available at any given time and are not guaranteed to be compatible with one another.

Example 1.1. Example Request URL (contract version in bold)

```
https://dfw.blockstorage.api.openstackcloud.com/v1/1234/
```

Note

This document pertains to contract version 1.0.

1.4. Request/Response Types

The Block Storage API supports both the JSON and XML data serialization formats. The request format is specified using the `Content-Type` header and is required for calls that have a request body. The response format can be specified in requests either by using the `Accept` header or by adding an `.xml` or `.json` extension to the request URI. Note that it is possible for a response to be serialized using a format different from the request. If no response format is specified, JSON is the default. If conflicting formats are specified using both an `Accept` header and a query extension, the query extension takes precedence.

Table 1.1. Response Formats

Format	Accept Header	Query Extension	Default
JSON	application/json	.json	Yes
XML	application/xml	.xml	No

In the request example below, notice that *`Content-Type`* is set to *`application/json`*, but *`application/xml`* is requested via the *`Accept`* header:

Example 1.2. Request with Headers (Getting Volume Types)

```
GET /v1/441446/types HTTP/1.1
Host: dfw.blockstorage.api.openstackcloud.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
Accept: application/xml
```

Therefore an XML response format is returned:

Example 1.3. Response with Headers

```
HTTP/1.1 200 OK
Date: Fri, 20 Jul 2012 20:32:13 GMT
Content-Length: 187
Content-Type: application/xml
X-Compute-Request-Id: req-8e0295cd-a283-46e4-96da-cae05cbfd1c7
```

```
<?xml version='1.0' encoding='UTF-8'?>
<volume_types>
    <volume_type id="1" name="SATA">
        <extra_specs/>
    </volume_type>
    <volume_type id="2" name="SSD">
        <extra_specs/>
    </volume_type>
</volume_types>
```

1.5. Limits

All accounts, by default, have a preconfigured 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.

1.5.1. Absolute Limits

Refer to the following table for the absolute limits that are set.

Table 1.2. Absolute Limits

Name	Description	Limit
Block Storage	Maximum amount of block storage (in gigabytes)	1 TB

1.6. Date/Time Format

The Block Storage Service uses an ISO-8601 compliant date format for the display and consumption of date/time values.

Example 1.4. DB Service Date/Time Format

```
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

See the table below for a description of the date/time format codes.

May 19th, 2011 at 8:07:08 AM, GMT-5 would have the following format:

```
2011-05-19T08:07:08-05:00
```

Table 1.3. Explanation of Date/Time Format Codes

Code	Description
yyyy	Four digit year
MM	Two digit month
dd	Two digit day of month
T	Separator for date/time
HH	Two digit hour of day (00-23)
mm	Two digit minutes of hour
ss	Two digit seconds of the minute
SSS	Three digit milliseconds of the second
Z	RFC-822 timezone

1.7. Faults

When an error occurs, the Block Storage Service returns a fault object containing an HTTP error response code that denotes the type of error. In the body of the response, the system will return additional information about the fault.

The following table lists possible fault types with their associated error codes and descriptions.

Fault Type	Associated Error Code	Description
badRequest	400	There was one or more errors in the user request.
unauthorized	401	The supplied token is not authorized to access the resources, either it's expired or invalid.
forbidden	403	Access to the requested resource was denied.
itemNotFound	404	The back-end services did not find anything matching the Request-URI.
badMethod	405	The request method is not allowed for this resource.
overLimit	413	Either the number of entities in the request is larger than allowed limits, or the user has exceeded allowable request rate limits. See the details element for more specifics. Contact support if you think you need higher request rate limits.
badMediaType	415	The requested content type is not supported by this service.
unprocessableEntity	422	The requested resource could not be processed on at the moment.
instanceFault	500	This is a generic server error and the message contains the reason for the error. This error could wrap several error messages and is a catch all.
notImplemented	501	The requested method or resource is not implemented.
serviceUnavailable	503	The Block Storage Service is not available.

The following two instanceFault examples show errors when the server has erred or cannot perform the requested operation:

Example 1.5. Example instanceFault Response: XML

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/xml
Content-Length: 121
Date: Mon, 28 Nov 2011 18:19:37 GMT

<instanceFault code="500" xmlns="http://docs.rackspace.com/cbs/api/v1.0">
    <message>
        The server has either erred or is incapable of performing the
 requested operation.
    </message>
</instanceFault>
```

Example 1.6. Example Fault Response: JSON

```
HTTP/1.1 500 Internal Server Error
Content-Length: 120
Content-Type: application/json; charset=UTF-8
Date: Tue, 29 Nov 2011 00:33:48 GMT

{
    "instanceFault": {
        "code": 500,
        "message": "The server has either erred or is incapable of performing
 the requested operation."
    }
}
```

The error code (`code`) is returned in the body of the response for convenience. The `message` element returns a human-readable message that is appropriate for display to the end user. The `details` element is optional and may contain information that is useful for tracking down an error, such as a stack trace. The `details` element may or may not be appropriate for display to an end user, depending on the role and experience of the end user.

The fault's root element (for example, `instanceFault`) may change depending on the type of error.

The following two `badRequest` examples show errors when the volume size is invalid:

Example 1.7. Example badRequest Fault on Volume Size Errors: XML

```
HTTP/1.1 400 None
Content-Type: application/xml
Content-Length: 121
Date: Mon, 28 Nov 2011 18:19:37 GMT

<badRequest code="400" xmlns="http://docs.rackspace.com/cbs/api/v1.0">
    <message>
        Volume 'size' needs to be a positive integer value, -1.0 cannot be
 accepted.
    </message>
</badRequest>
```

Example 1.8. Example badRequest Fault on Volume Size Errors: JSON

```
HTTP/1.1 400 None
Content-Length: 120
Content-Type: application/json; charset=UTF-8
Date: Tue, 29 Nov 2011 00:33:48 GMT

{
    "badRequest": {
        "code": 400,
        "message": "Volume 'size' needs to be a positive integer value, -1.0
 cannot be accepted."
    }
}
```

The next two examples show `itemNotFound` errors:

Example 1.9. Example itemNotFound Fault: XML

```
HTTP/1.1 404 Not Found
Content-Length: 147
Content-Type: application/xml; charset=UTF-8
Date: Mon, 28 Nov 2011 19:50:15 GMT

<itemNotFound code="404" xmlns="http://docs.rackspace.com/cbs/api/v1.0">
    <message>
        The resource could not be found.
    </message>
</itemNotFound>
```

Example 1.10. Example itemNotFound Fault: JSON

```
HTTP/1.1 404 Not Found
Content-Length: 78
Content-Type: application/json; charset=UTF-8
Date: Tue, 29 Nov 2011 00:35:24 GMT

{
    "itemNotFound": {
        "code": 404,
        "message": "The resource could not be found."
    }
}
```

1.8. Volume Status

When making an API call to create, list, or delete volume(s), the following volume status values are possible:

- CREATING – The volume is being created.

- AVAILABLE – The volume is read to be attached to an instance.
- ATTACHING – The volume is attaching to an instance.
- IN-USE – The volume is attached to an instance.
- DELETING – The volume is being deleted.
- ERROR – There has been some error with the volume.
- ERROR_DELETING – There was an error deleting the volume.

2. API Operations

The Volume API allows you to manage volumes and snapshots that can be used with the Compute API.

2.1. Volumes

Verb	URI	Description
POST	`/volumes?volume=&size=& display_description=& display_name=&metadata=& snapshot_id=&volume_type=`	Creates the volume.
GET	`/volumes`	View a list of volumes.
GET	`/volumes/{volume_id}`	View all information about a single Volume.
DELETE	`/volumes/{volume_id}`	Delete a single Volume.

2.1.1. Create Volume

Verb	URI	Description
POST	`/volumes?volume=&size=& display_description=& display_name=&metadata=& snapshot_id=&volume_type=`	Creates the volume.

Normal Response Code(s): 200

The below example creates a SATA volume called "vol-001" that has a size of 30 GB.

Table 2.1. Create Volume Request Parameters

Name	Style	Type	Description
`volume`	Query		A partial representation of a volume used in the creation process. The `volume` parameter should always be supplied.
`size`	Query		The size (in GB) of the volume. The `size` parameter should always be supplied.
`display_description`	Query		A description of the volume. The `display_description` parameter is optional.
`display_name`	Query		The name of the volume. The `display_name` parameter is optional.
`metadata`	Query		Key and value metadata pairs defined by the admin. The `metadata` parameter is optional.
`snapshot_id`	Query		The optional snapshot from which to create a volume. The `snapshot_id` parameter is optional.
`volume_type`	Query		The type of volume to create, either SATA or SSD. If not defined, then the default, SATA, is used. The `volume_type` parameter is optional.

This operation does not require a request body.

Note that you use the `os-volume_attachments` API call (/servers/{server_id}/os-volume_attachments) to attach the new volume to your Server (with the specified {server_id}).

2.1.2. List Volumes

Verb	URI	Description
GET	`/volumes`	View a list of volumes.

Normal Response Code(s): 200

This operation does not require a request body.

2.1.3. Show Volume

Verb	URI	Description
GET	`/volumes/{volume_id}`	View all information about a single Volume.

Normal Response Code(s): 200

2.1.4. Delete Volume

Verb	URI	Description
DELETE	`/volumes/{volume_id}`	Delete a single Volume.

Normal Response Code(s): 202

This operation does not require a request body and does not return a response body.

2.2. Volume types

Verb	URI	Description
GET	`/types`	Request a list of Volume Types.
GET	`/types/{volume_type_id}`	Request a single Volume Type.

2.2.1. List Volume Types

Verb	URI	Description
GET	`/types`	Request a list of Volume Types.

Normal Response Code(s): 200

This operation does not require a request body.

Note that two storage types are currently supported: SATA and SSD. SATA is the standard performance storage option and SSD is the high performance option.

2.2.2. Describe Volume Type

Verb	URI	Description
GET	`/types/{volume_type_id}`	Request a single Volume Type.

Normal Response Code(s): 200

This operation does not require a request body.

2.3. Snapshots

Verb	URI	Description
POST	`/snapshots?snapshot=&volume_id=&force=&display_name=&display_description=`	Create a Snapshot.
GET	`/snapshots`	View a list of simple Snapshot entities.
GET	`/snapshots/{snapshot_id}`	View all information about a single Snapshot.
DELETE	`/snapshots/{snapshot_id}`	Delete a single Snapshot.

2.3.1. Create Snapshot

Verb	URI	Description
POST	`/snapshots?snapshot=&volume_id=&force=&display_name=&display_description=`	Create a Snapshot.

Normal Response Code(s): 201

Creating a snapshot makes a point-in-time copy of the volume. All writes to the volume should be flushed *before* creating the snapshot, either by un-mounting any file systems on the volume, or by detaching the volume before creating the snapshot. Snapshots are incremental, so each time you create a new snapshot, you are appending the incremental changes for the new snapshot to the previous one. The previous snapshot is still available. Note that you can create a new volume from the snapshot if desired.

Table 2.2. Create Snapshot Request Parameters

Name	Style	Type	Description
`snapshot`	Query		A partial representation of a snapshot used in the creation process. The `snapshot` parameter should always be supplied.
`volume_id`	Query		The ID of the volume to snapshot. The `volume_id` parameter should always be supplied.
`force`	Query		[True/False] Indicate whether to snapshot, even if the volume is attached. Default==False. The `force` parameter is optional.
`display_name`	Query		Name of the snapshot. Default==None. The `display_name` parameter is optional.
`display_description`	Query		Description of snapshot. Default==None. The `display_description` parameter is optional.

This operation does not require a request body.

2.3.2. List Snapshots

Verb	URI	Description
GET	`/snapshots`	View a list of simple Snapshot entities.

Normal Response Code(s): 200

This operation does not require a request body.

2.3.3. Show Snapshot

Verb	URI	Description
GET	`/snapshots/{snapshot_id}`	View all information about a single Snapshot.

Normal Response Code(s): 200

This operation does not require a request body.

2.3.4. Delete Snapshot

Verb	URI	Description
DELETE	`/snapshots/{snapshot_id}`	Delete a single Snapshot.

Normal Response Code(s): 202

This operation does not require a request body and does not return a response body.