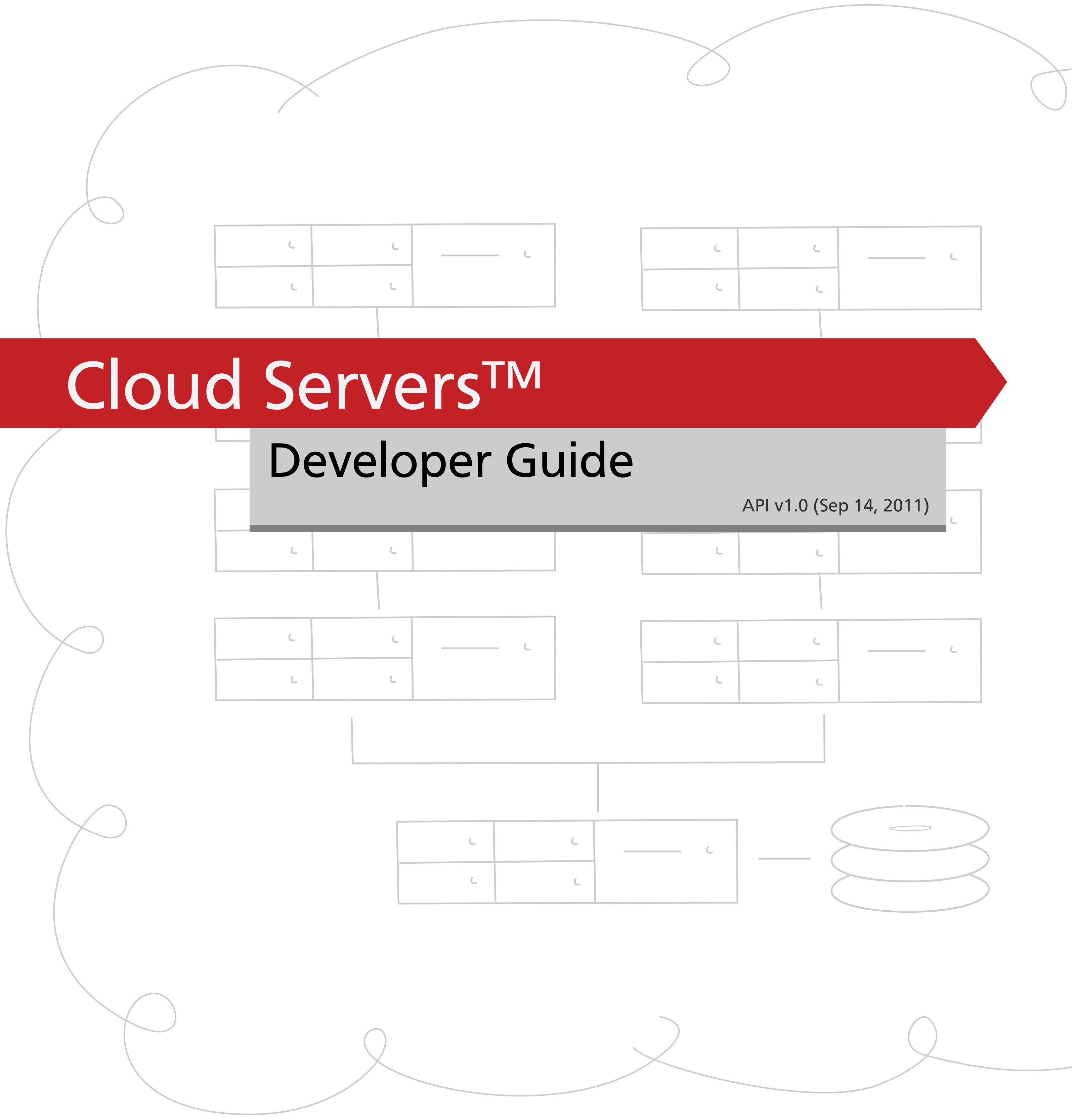

Cloud Servers™

Developer Guide

API v1.0 (Sep 14, 2011)

rackspace®
HOSTING

docs.rackspace.com/api

Cloud Servers™ Developer Guide

API v1.0 (2011-09-14)

Table of Contents

List of Tables

List of Examples

1. Overview

Rackspace Cloud Servers™ is a compute service that provides server capacity in the cloud. Cloud Servers come in different flavors of memory, disk space, and CPU, and can be provisioned in minutes. There are no contracts or commitments. You may use a Cloud Server for as long or as little as you choose. You pay only for what you use and pricing is by the hour. Interactions with Rackspace Cloud Servers can occur via the Rackspace Cloud Control Panel (GUI) or programmatically via the Rackspace Cloud Servers API.

We welcome feedback, comments, and bug reports at support@rackspacecloud.com.

1.1. Intended Audience

This Guide is intended to assist software developers who want to develop applications using the Rackspace Cloud Servers API. To use the information provided here, you should first have a general understanding of the Rackspace Cloud Servers service and have access to an active Rackspace Cloud Servers account. You should also be familiar with:

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

1.2. 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
Mar 20, 2012	• Added note to get image details about missing serverId field in the response body.
Sep 14, 2011	• Added cURL example in the Authentication section.
Mar 10, 2011	• Fixed error referring to X-Auth-User instead of X-Auth-Key.
Jan 12, 2011	• Added section numbers.
Jan 4, 2011	• Expanded authentication information for UK release.
Oct 15, 2009	• Added note stating that *`changes-since`* returns deleted items on server lists. • Added itemNotFound as a possible fault when creating a server. • Removed statement saying that creating an image with the same name as another will replace the original image.
Sep 16, 2009	• Updated description for setting/getting backup schedules. • Increased **POST**s to /servers to 50 per day. • Added server status `RESIZE`. • Mentioned trailing slash on version request. • Removed statement that image ID is optional on a rebuild. • Mentioned that a **DELETE** on image may return a 204. • Fixed server update call to return a 204 rather than a 202.
Aug 11, 2009	• Clarified DEPRECATED in version call. • Changed "delete" backup schedule to "disable" backup schedule. • Added **DELETE** operation on images. • Removed mention of CPU time in flavors. • Mentioned that all methods may return an overLimit fault. • Fixed examples for authentication, server reboot, limits, and version calls.
Jul 14, 2009	• Initial release.

1.3. Additional Resources

You can download the most current version of this document from the Rackspace Cloud website at http://docs.rackspacecloud.com/servers/api/cs-devguide-latest.pdf.

For more details about the Cloud Servers service, please refer to http://www.rackspacecloud.com/cloud_hosting_products/servers. Related documents, including an API Language Binding Guide, are available at the same site, as are links to Rackspace's official support channels, including knowledge base articles, forums, phone, chat, and email.

You can also follow updates and announcements via twitter at http://www.twitter.com/rackcloud

2. Concepts

To use the Cloud Servers API effectively, you should understand several key concepts:

2.1. Server

A server is a virtual machine instance in the Cloud Servers system. Flavor and image are requisite elements when creating a server.

2.2. Flavor

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

2.3. Image

An image is a collection of files used to create or rebuild a server. Rackspace provides 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.

2.4. Backup Schedule

A backup schedule can be defined to create server images at regular intervals (daily and weekly). Backup schedules are configurable per server.

2.5. Reboot

The reboot function allows for 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.

2.6. Rebuild

The rebuild function removes all data on the server and replaces it with the specified image. Server ID and IP addresses remain the same.

2.7. Resize

The resize function converts 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 they are not confirmed or reverted.

2.8. Shared IP Address

Public IP addresses can be shared across multiple servers for use in various high availability scenarios. When an IP address is shared to another server, the cloud network restrictions are modified to allow each server to listen to and respond on that IP address (you may optionally specify that the target server network configuration be modified). Shared IP addresses can be used with many standard heartbeat facilities (e.g. keepalived) that monitor for failure and manage IP failover.

2.9. Shared IP Group

A shared IP group is a collection of servers that can share IPs with other members of the group. Any server in a group can share one or more public IPs with any other server in the group. With the exception of the first server in a shared IP group, servers must be launched into shared IP groups. A server may only be a member of one shared IP group.

3. General API Information

The Cloud Servers API is implemented using a ReSTful web service interface. Like other products in the Rackspace Cloud suite, Cloud Servers shares a common token authentication system that allows seamless access between products and services.

Note

All requests to authenticate and operate against Cloud Servers are performed using SSL over HTTP (HTTPS) on TCP port 443.

3.1. Authentication

Each ReST request against the Cloud Servers system requires the inclusion of a specific authorization token HTTP x-header, defined as `X-Auth-Token`. Clients obtain this token, along with the Cloud Servers API URL, by first using the Rackspace Cloud Authentication Service and supplying a valid username and API access key.

The Rackspace Cloud Authentication Service is a ReSTful web service. It is the entry point to all Rackspace Cloud APIs.

To access the Authentication Service, you must know whether your account is US-based or UK-based:

- US-based accounts authenticate through https://auth.api.rackspacecloud.com/v1.0.
- UK-based accounts authenticate through https://lon.auth.api.rackspacecloud.com/v1.0.

Your account may be based in either the US or the UK; this is not determined by your physical location but by the location of the Rackspace retail site which was used to create your account:

- If your account was created via http://www.rackspacecloud.com, it is a US-based account.
- If your account was created via http:/www.rackspace.co.uk, it is a UK-based account.

If you are unsure how your account was created, use the Rackspace contact information at either site to ask for help.

3.1.1. Request

To authenticate, you must supply your username and API access key in x-headers:

- Use your Rackspace Cloud username as the username for the API. Place it in the `X-Auth-User` x-header.
- Obtain your API access key from the Rackspace Cloud Control Panel in the Your Account | API Access section. Place it in the `X-Auth-Key` x-header.

Example 3.1. Authentication Request (US-Based Account)

```
GET /v1.0 HTTP/1.1
Host: auth.api.rackspacecloud.com
X-Auth-User: jdoe
X-Auth-Key: a86850deb2742ec3cb41518e26aa2d89
```

3.1.2. Response

The Cloud Servers API may return any of the HTTP/1.1 response codes defined by RFC-2616 Section 10. If authentication is successful, an HTTP status 204 (No Content) is returned with three cloud service headers, `X-Server-Management-Url`, `X-Storage-Url`, `X-CDN-Management-Url`, as well as `X-Auth-Token`. An HTTP status of 401 (Unauthorized) is returned if authentication fails. All operations against Cloud Servers should be performed against the URL specified in `X-Server-Management-Url` (which is dynamic and subject to change) and must include the `X-Auth-Token` header as noted above. The URLs specified in `X-Storage-Url` and `X-CDN-Management-Url` are specific to the Cloud Files product and may be ignored for purposes of interacting with Cloud Servers.

Example 3.2. Authentication Response

```
HTTP/1.1 204 No Content
Date: Mon, 12 Nov 2007 15:32:21 GMT
Server: Apache
X-Server-Management-Url: https://servers.api.rackspacecloud.com/v1.0/35428
X-Storage-Url: https://storage.clouddrive.com/v1/CloudFS_9c83b-5ed4
X-CDN-Management-Url: https://cdn.clouddrive.com/v1/CloudFS_9c83b-5ed4
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

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

Note that API operation URIs specified throughout this document are relative, this is, they should be appended to the end of the `X-Server-Management-Url` that is returned from the authentication system. For example, in the sample response above, you would list servers by performing a **GET** against https://servers.api.rackspacecloud.com/v1.0/35428/servers.

3.1.3. cURL Example

The following example demonstrates how to use cURL to obtain the authorization token and the URL of the storage system. This example uses the US-based URL https://auth.api.rackspacecloud.com/v1.0. UK-based accounts authenticate through https://lon.auth.api.rackspacecloud.com/v1.0.

```
curl -D - \
-H "X-Auth-Key: a86850deb2742ec3cb41518e26aa2d89" \
-H "X-Auth-User: jdoe" \
https://auth.api.rackspacecloud.com/v1.0

HTTP/1.1 204 No Content
```

```
Date: Thu, 09 Jul 2009 15:31:39 GMT
Server: Apache/2.2.3
X-Storage-Url: https://storage.clouddrive.com/v1/CF_xer7_343
X-CDN-Management-Url: https://cdn.clouddrive.com/v1/CF_xer7_343
X-Auth-Token: fc81aaa6-98a1-9ab0-94ba-aba9a89aa9ae
Content-Length: 0
Connection: close
Content-Type: application/octet-stream
```

The storage URL, CDN management URL, and authentication token are returned in the headers of the response. After authentication, you can use cURL to perform **GET**, **DELETE**, **POST** and **PUT** requests on the storage and CDN services.

While an authentication token lasts, you can continue to perform requests, but once a token expires it returns an HTTP error code 401 Unauthorized. Given that an `X-Auth-Token` is good for 24 hours, long running or high request rate jobs should not try to authenticate at api.auth.rackspace.com on every request. You don't need to request another `X-Auth-Token` again until your existing `X-Auth-Token` expires. At that point you must obtain another authorization token. As a best practice example, here is some pseudo-code for re-authenticating. The best scalable process flow would be:

1. Begin requests by going to auth.api.rackspace.com for an `X-Auth-Token`.
2. Send request `X-Storage-URL` using the `X-Auth-Token` obtained in Step 1.
3. Repeat step 2 using the same `X-Auth-Token` retrieved in Step 1 until either the job finishes or you get a result code of 401 (Unauthorized) .
    - If the job finishes, you can allow the token to expire with no further action.
    - If result code is 401 then send a request to auth.api.rackspacecloud.com for a new `X-Auth-Token`.

3.2. Request/Response Types

The Cloud Servers API supports both the JSON and XML data serialization formats. The request format is specified using the `Content-Type` header and is required for operations that have a request body. The response format can be specified in requests using either the `Accept` header or 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 (see example below). 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 3.1. JSON and XML Response Formats

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

Example 3.3. Request with Headers: JSON

```
POST /v1.0/214412/images HTTP/1.1
Host: servers.api.rackspacecloud.com
Content-Type: application/json
Accept: application/xml
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
{
    "image" : {
        "serverId" : 12,
        "name" : "Just in case"}
}
```

Example 3.4. Response with Headers: XML

```
HTTP/1.1 200 OKAY
Date: Mon, 12 Nov 2007 15:55:01 GMT
Server: Apache
Content-Length: 185
```

```
Content-Type: application/xml; charset=UTF-8
<image xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
       id="22" name="Just in case" serverId="12"
       created="2010-10-10T12:00:00Z"
       status="SAVING" progress="0" />
```

Notice, in the above example, that the content type is set to application/json but application/xml is requested via the `Accept` header. An alternative method of achieving the same result is illustrated below – this time we utilize a URI extension instead of an `Accept` header.

Example 3.5. Request with Extension: JSON

```
POST /v1.0/214412/images .xml HTTP/1.1
Host: servers.api.rackspacecloud.com
Content-Type: application/json
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

```
{
    "image" : {
        "serverId" : 12,
        "name" : "Just in case"}
}
```

3.3. Content Compression

Request and response body data may be encoded with gzip compression in order to accelerate interactive performance of API calls and responses. This is controlled using the `Accept-Encoding` header on the request from the client and indicated by the `Content-Encoding` header in the server response. Unless the header is explicitly set, encoding defaults to disabled.

Table 3.2. Encoding Headers

Header Type	Name	Value
HTTP/1.1 Request	`Accept-Encoding`	gzip
HTTP/1.1 Response	`Content-Encoding`	gzip

3.4. Chunked Transfer Encoding

The Cloud Server API does not support chunked transfer-encoding.

3.5. Persistent Connections

By default, the API supports persistent connections via HTTP/1.1 keepalives. All connections will be kept alive unless the connection header is set to close.

To prevent abuse, HTTP sessions have a timeout of 20 seconds before being closed.

Note

The server may close the connection at any time and clients should not rely on this behavior.

3.6. Paginated Collections

To reduce load on the service, list operations will return a maximum of 1,000 items at a time. To navigate the collection, the parameters *`limit`* and *`offset`* can be set in the URI (e.g.?*`limit`*=0&*`offset`*=0). If an offset is given beyond the end of a list an empty list will be returned. Note that list operations never return itemNotFound (404) faults.

3.7. Caching

The Cloud Servers API makes extensive use of caching layers at various tiers of the system. Purging mechanisms exist to ensure that objects served out of cache are accurate and up to date. **GET**s returning a cached entity return a 203 (Cached) to signal users that the value is being served out of cache. Additionally, cached entities have the following header set:

Table 3.3. Last Modified Header

Header	Description
`Last-Modified`	Date and time when the entity was last updated.

3.8. 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 Unix time (the number of seconds since January 1, 1970, 00:00:00 UTC, not counting leap seconds). If nothing has changed since the *`changes-since`* time, a 304 (Not Modified) response will be returned. If data has changed, only the items changed since the specified time will be returned in the response. For example, performing a **GET** against https://api.servers.rackspacecloud.com/v1.0/224532/servers?*`changes-since`*=1244012982 would list all servers that have changed since Wed, 03 Jun 2009 07:09:42 UTC.

3.9. 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.

Note

If the default limits are too low for your particular application, please contact Rackspace Cloud support to request an increase. All requests require reasonable justification.

3.9.1. Rate Limits

We specify rate limits in terms of both a human-readable wild-card URI and a machine-processable regular expression. The regular expression boundary matcher '^' 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.rackspacecloud.com/v1.0/3542812**/servers**.

Table 3.4. Default Rate Limits

Verb	URI	RegEx	Default
POST	*	.*	10/min
POST	*/servers	^/servers	50/day
PUT	*	.*	10/min
GET	*changes-since*	changes-since	3/min
DELETE	*	.*	100/min

Rate limits are applied in order relative to the verb, going from least to most specific. For example, although the threshold for **POST** to */servers is 50 per day, one cannot **POST** to */servers more than 10 times within a single minute because the rate limits for any **POST** is 10/min.

In the event you exceed the thresholds established for your account, a 413 (Rate Control) HTTP response will be returned with a `Reply-After` header to notify the client when they can attempt to try again.

3.9.2. Absolute Limits

“Maximum total amount of RAM (GB)” limits the number of instances you can run based on the total aggregate RAM size. For example, with the default limit of 50GB, you may launch a maximum of 200 256MB cloud servers, or 100 512MB cloud servers, or 50 1GB cloud servers, or 20 2GB + 40 256MB cloud servers, etc. These limits apply to creating as well as resizing servers.

Table 3.5. Default Absolute Limits

Limit	Default
Maximum total amount of RAM (GB)	50
Maximum number of shared IP groups	25
Maximum number of members per shared IP group	25

3.9.3. Determining Limits Programmatically

Applications can programmatically determine current account limits using the /limits URI as follows:

Verb	URI	Description
GET	/limits	Returns the current limits for the account

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation does not require a request body.

Example 3.6. Limit Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<limits xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <rate>
     <limit verb="POST"  URI="*" regex=".*"
            value="10" remaining="2"
     unit="MINUTE" resetTime="1244425439" />

     <limit verb="POST" URI="*/servers" regex="^/servers"
            value="25" remaining="24"
     unit="DAY" resetTime="1244511839" />

     <limit verb="PUT"  URI="*" regex=".*"
            value="10" remaining="2"
     unit="MINUTE" resetTime="1244425439" />

     <limit verb="GET"  URI="*" regex=".*"
            value="3" remaining="3"
     unit="MINUTE" resetTime="1244425439" />

     <limit verb="DELETE"  URI="*" regex=".*"
            value="100" remaining="100"
     unit="MINUTE" resetTime="1244425439" />
  </rate>
  <absolute>
    <limit name="maxTotalRAMSize" value="51200" />
    <limit name="maxIPGroups"   value="50" />
    <limit name="maxIPGroupMembers" value="25" />
  </absolute>
</limits>
```

Example 3.7. Limit Response: JSON

```
{
    "limits" : {
        "rate" : [
            {
                "verb" : "POST",
                "URI" : "*",
                "regex" : ".*",
                "value" : 10,
                "remaining" : 2,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            },
            {
                "verb" : "POST",
                "URI" : "*/servers",
                "regex" : "^/servers",
                "value" : 25,
                "remaining" : 24,
                "unit" : "DAY",
                "resetTime" : 1244511839
            },
            {
                "verb" : "PUT",
                "URI" : "*",
                "regex" : ".*",
                "value" : 10,
                "remaining" : 2,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            },
            {
                "verb" : "GET",
                "URI" : "*",
                "regex" : ".*",
                "value" : 3,
                "remaining" : 3,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            },
            {
                "verb" : "DELETE",
                "URI" : "*",
                "regex" : ".*",
                "value" : 100,
                "remaining" : 100,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            }
        ],
        "absolute" : {
            "maxTotalRAMSize" : 51200,
            "maxIPGroups" : 50,
            "maxIPGroupMembers" : 25
        }
    }
}
```

```
{
    "limits" : {
        "rate" : [
            {
                "verb" : "POST",
                "URI" : "*",
                "regex" : ".*",
                "value" : 10,
                "remaining" : 2,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            },
            {
                "verb" : "POST",
                "URI" : "*/servers",
                "regex" : "^/servers",
                "value" : 25,
                "remaining" : 24,
                "unit" : "DAY",
                "resetTime" : 1244511839
            },
            {
                "verb" : "PUT",
                "URI" : "*",
                "regex" : ".*",
                "value" : 10,
                "remaining" : 2,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            },
            {
                "verb" : "GET",
                "URI" : "*",
                "regex" : ".*",
                "value" : 3,
                "remaining" : 3,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            },
            {
                "verb" : "DELETE",
                "URI" : "*",
                "regex" : ".*",
                "value" : 100,
                "remaining" : 100,
                "unit" : "MINUTE",
                "resetTime" : 1244425439
            }
        ],
        "absolute" : {
            "maxTotalRAMSize" : 51200,
            "maxIPGroups" : 50,
            "maxIPGroupMembers" : 25
        }
    }
}
```

3.10. API Version

The Cloud Servers API uses a URI versioning scheme. The first element of the path contains the target version identifier (e.g. https://servers.api.rackspacecloud.com/ v1.0/...) All requests (except to query for version — see below) must contain a target version. New features and functionality that do not break API-compatibility will be introduced in the current version of the API and the URI will remain unchanged. Any features or functionality changes that would necessitate a break in API-compatibility will require a new version, which will result in the URI version being updated accordingly. When new API versions are released, older versions will be marked as DEPRECATED. Rackspace will 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.

Example 3.8. Versions List Request

```
GET HTTP/1.1
Host: servers.api.rackspacecloud.com
```

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 413, 500, 503

This operation does not require a request body.

Example 3.9. Versions List Response: XML

```
<versions xmlns="http://docs.rackspacecloud.com/servers/api/v1.0" >
   <version status="CURRENT" id="v1.0"/>
   <version status="BETA" id="v1.1"/>
</versions>
```

Example 3.10. Versions List Response: JSON

```
{"versions": [{
   "status": "CURRENT",
   "id": "v1.0"},
   {
   "status": "BETA",
   "id": "v1.1"
}]}
```

You can also obtain additional information about a specific version by performing a **GET** on the base version URL (e.g. https://servers.api.rackspacecloud.com/v1.0/). Version request URLs should always end with a trailing slash (/). If the slash is omitted, the server may respond with a 302 redirection request. Format extensions may be placed after the slash (e.g. https://servers.api.rackspacecloud.com/v1.0/.xml). Note that this is a special case that does not hold true for other API requests. In general, requests such as /servers.xml and /servers/.xml are handled equivalently.

Example 3.11. Version Details Request

```
GET HTTP/1.1
Host: servers.api.rackspacecloud.com/v1.0/
```

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit(413)

This operation does not require a request body

Example 3.12. Version Details Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<version xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  id="v1.0"
  status="BETA"
  docURL="http://docs.rackspacecloud.com/cs/cs-devguid-v1.0.pdf"
  wadl="https://servers.api.rackspacecloud.com/v1.0/application.wadl"/>
```

Example 3.13. Version Details Response: JSON

```
{
    "version": {
        "status": "BETA",
        "id": "v1.0",
        "docURL" : "http://docs.rackspacecloud.com/cs/cs-devguid-v1.0.pdf",
        "wadl" : "https://servers.api.rackspacecloud.com/v1.0/application.
wadl"
    }
}
```

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).

Note

If there is a discrepancy between the two specifications, the WADL is authoritative as it contains the most accurate and up-to-date description of the API service.

3.11. Faults

When an error occurs, the system will return an HTTP error response code denoting the type of error. The system will also return additional information about the fault in the body of the response.

Example 3.14. Fault Response: XML

```
<cloudServersFault xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 code="500">
  <message>Fault!</message>
  <details>Error Details...</details>
</cloudServersFault>
```

Example 3.15. Fault Response: JSON

```
{
    "cloudServersFault" : {
        "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 may or may not be appropriate for display to an end user.

The root element of the fault (e.g. cloudServersFault) may change depending on the type of error. The following is a list of possible elements along with their associated error codes.

Table 3.6. Fault Elements and Error Codes

Fault Element	Associated Error Codes	Expected in All Requests?
cloudServersFault	500, 400, other codes possible	✓
serviceUnavailable	503	✓
unauthorized	401	✓
badRequest	400	✓
overLimit	413	✓
badMediaType	415	
badMethod	405	
itemNotFound	404	
buildInProgress	409	
serverCapacityUnavailable	503	
backupOrResizeInProgress	409	
resizeNotAllowed	403	
notImplemented	501	

Example 3.16. Fault Response, Item Not Found: XML

```
<itemNotFound xmlns="http://docs.rackspacecloud.com/servers/api/v1.0" code=
"404">
  <message>Not Found</message>
  <details>Error Details...</details>
</itemNotFound>
```

Example 3.17. Fault Response, Item Not Found: JSON

```
{
    "itemNotFound" : {
        "code" : 404,
        "message" : "Not Found",
        "details" : "Error Details..."
    }
}
```

From an XML schema perspective, all API faults are extensions of the base fault type CloudServersAPIFault. When working with a system that binds XML to actual classes (such as JAXB), one should be capable of using CloudServersAPIFault as a ?catch-all? if there's no interest in distinguishing between individual fault types.

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

Example 3.18. Fault Response, Over Limit: XML

```
<overLimit xmlns="http://docs.rackspacecloud.com/servers/api/v1.0" code="413"
    retryAfter="2010-08-01T00:00:00Z">
  <message>OverLimit Retry...</message>
  <details>Error Details...</details>
</overLimit>
```

Example 3.19. Fault Response, Over Limit: JSON

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

4. API Operations

4.1. Servers

4.1.1. List Servers

Verb	URI	Description
GET	/servers	List all servers (IDs and names only)
GET	/servers/detail	List all servers (all details)

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation provides a list of servers associated with your account. Servers that have been deleted are not included in this list. Servers contain a status attribute that can be used as an indication of the current server state. Servers with an `ACTIVE` status are available for use. Other possible values for the status attribute include: `BUILD`, `REBUILD`, `SUSPENDED`, `QUEUE_RESIZE`, `PREP_RESIZE`, `RESIZE`, `VERIFY_RESIZE`, `PASSWORD`, `RESCUE`, `REBOOT`, `HARD_REBOOT`, `SHARE_IP`, `SHARE_IP_NO_CONFIG`, `DELETE_IP`, and `UNKNOWN`.

When retrieving a list of servers via the changes-since parameter (see Efficient Polling with the Changes-Since Parameter), the list will contain servers that have been deleted since the changes-since time.

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

Note

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

This operation does not require a request body.

Example 4.1. Servers List Response: XML (detail)

```
<?xml version="1.0" encoding="UTF-8"?>

<servers xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">

  <server id="1234" name="sample-server"
   imageId="1" flavorId="1"
   status="BUILD" progress="60"
   hostId="e4d909c290d0fb1ca068ffaddf22cbd0"
   >
    <metadata>
```

```
      <meta key="Server Label">Web Head 1</meta>
      <meta key="Image Version">2.1</meta>
    </metadata>
    <addresses>
      <public>
          <ip addr="67.23.10.132"/>
          <ip addr="67.23.10.131"/>
      </public>
      <private>
          <ip addr="10.176.42.16"/>
      </private>
    </addresses>
  </server>

  <server id="5678" name="sample-server2"
   imageId="1" flavorId="1"
   status="ACTIVE"
   hostId="9e107d9d372bb6826bd81d3542a419d6">
    <metadata>
      <meta key="Server Label">DB 1</meta>
    </metadata>
    <addresses>
      <public>
          <ip addr="67.23.10.133"/>
      </public>
      <private>
          <ip addr="10.176.42.17"/>
      </private>
    </addresses>
  </server>

</servers>
```

Example 4.2. Servers List Response: JSON (detail)

```
{
    "servers" : [
        {
            "id" : 1234,
            "name" : "sample-server",
            "imageId" : 1,
            "flavorId" : 1,
            "hostId" : "e4d909c290d0fb1ca068ffaddf22cbd0",
            "status" : "BUILD",
            "progress" : 60,
            "addresses" : {
                "public" : [
                    "67.23.10.132",
                    "67.23.10.131"
                ],
                "private" : [
                    "10.176.42.16"
                ]
            },
            "metadata" : {
                "Server Label" : "Web Head 1",
                "Image Version" : "2.1"
            }
        },
        {
         "id" : 5678,
            "name" : "sample-server2",
            "imageId" : 1,
            "flavorId" : 1,
            "hostId" : "9e107d9d372bb6826bd81d3542a419d6",
            "status" : "ACTIVE",
            "addresses" : {
                "public" : [
                    "67.23.10.133"
                ],
                "private" : [
                    "10.176.42.17"
                ]
            },
            "metadata" : {
                "Server Label" : "DB 1"
            }
        }
    ]
}
```

4.1.2. Create Server

Verb	URI	Description
POST	/servers	Create a new server

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badMediaType(415), itemNotFound (404), badRequest (400), serverCapacityUnavailable (503), overLimit (413)

Status Transition:	BUILD → ACTIVE
	BUILD → ERROR (on error)

This 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 /server/ id , which will return a progress attribute (0-100% completion).

A password will be randomly generated for you and returned in the response object. For security reasons, it will not be returned in subsequent **GET** calls against a given server ID.

Custom cloud server metadata can also be supplied at launch time. This metadata is stored in the API system where it is retrievable by querying the API for server status. The maximum size of the metadata key and value is each 255 bytes and the maximum number of key-value pairs that can be supplied per server is 5.

You may further customize a cloud server by injecting data into the file system of the cloud server itself. This is useful, for example, for inserting ssh keys, setting configuration files, or storing data that you want to retrieve from within the instance itself. It is intended to provide a minimal amount of launch-time personalization. If significant customization is required, a custom image should be created. The max size of the file p ath data is 255 bytes while the max size of the file contents is 10KB. Note that the file contents should be encoded as a Base64 string and the 10KB limit refers to the number of bytes in the decoded data not the number of characters in the encoded data. The maximum number of file path/content pairs that can be supplied is 5. Any existing files that match the specified file will be renamed to include the extension bak followed by a time stamp. For example, the file /etc/passwd will be backed up as /etc/passwd.bak.1246036261.5785 . All files will have root and the root group as owner and group owner, respectively and will allow user and group read access only (`-r--r-----`).

Servers in the same shared IP group can share public IPs for various high availability and load balancing configurations. To launch an HA server, include the optional sharedIpGroupId element and the server will be launched into that shared IP group.

If you intend to use a shared IP on the server being created and have no need for a separate public IP address, you may launch the server into a shared IP group and specify an IP address from that shared IP group to be used as its public IP. You can accomplish this by specifying the public shared IP address in your request. This is optional and is only valid if sharedIpGroupId is also supplied.

Note

sharedIpGroupId is an optional parameter and for optimal performance, should ONLY be specified when intending to share IPs between servers.

Example 4.3. Server Create Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 name="new-server-test" imageId="1" flavorId="1">
  <metadata>
    <meta key="My Server Name">Apache1</meta>
  </metadata>
  <personality>
    <file path="/etc/banner.txt">
        ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBp
        dCBtb3ZlcyBpbiBqdXN0IHN1Y2ggYSBkaXJlY3Rpb24gYW5k
        IGF0IHN1Y2ggYSBzcGVlZC4uLkl0IGZlZWxzIGFuIGltcHVs
        c2lvbi4uLnRoaXMgaXMgdGhlIHBsYWNlIHRvIGdvIG5vdy4g
        QnV0IHRoZSBza3kga25vd3MgdGhlIHJlYXNvbnMgYW5kIHRo
        ZSBwYXR0ZXJucyBiZWhpbmQgYWxsIGNsb3VkcywgYW5kIHlv
        dSB3aWxsIGtub3csIHRvbywgd2hlbiB5b3UgbGlmdCB5b3Vy
        c2VsZiBoaWdoIGVub3VnaCB0byBzZWUgYmV5b25kIGhvcml6
        b25zLiINCg0KLVJpY2hhcmQgQmFjaA==
    </file>
  </personality>
</server>
```

Example 4.4. Server Create Request: JSON

```
{
    "server" : {
        "name" : "new-server-test",
        "imageId" : 1,
        "flavorId" : 1,
        "metadata" : {
            "My Server Name" : "Apache1"
        },
        "personality" : [
            {
                "path" : "/etc/banner.txt",
                "contents" : "ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBp
 dCBtb3ZlcyBpbiBqdXN0IHN1Y2ggYSBkaXJlY3Rpb24gYW5k
 IGF0IHN1Y2ggYSBzcGVlZC4uLkl0IGZlZWxzIGFuIGltcHVs
 c2lvbi4uLnRoaXMgaXMgdGhlIHBsYWNlIHRvIGdvIG5vdy4g
 QnV0IHRoZSBza3kga25vd3MgdGhlIHJlYXNvbnMgYW5kIHRo
 ZSBwYXR0ZXJucyBiZWhpbmQgYWxsIGNsb3VkcywgYW5kIHlv
 dSB3aWxsIGtub3csIHRvbywgd2hlbiB5b3UgbGlmdCB5b3Vy
 c2VsZiBoaWdoIGVub3VnaCB0byBzZWUgYmV5b25kIGhvcml6
 b25zLiINCg0KLVJpY2hhcmQgQmFjaA=="
            }
        ]
    }
}
```

Example 4.5. Server Create Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 id="1235" name="new-server-test"
 imageId="1" flavorId="1"
 hostId="e4d909c290d0fb1ca068ffaddf22cbd0"
 progress="0" status="BUILD"
 adminPass="GFf1j9aP">
  <metadata>
    <meta key="My Server Name">Apache1</meta>
  </metadata>
  <addresses>
    <public>
      <ip addr="67.23.10.138"/>
    </public>
    <private>
      <ip addr="10.176.42.19"/>
    </private>
  </addresses>
</server>
```

Example 4.6. Server Create Response: JSON

```
{
   "server" : {
     "id" : 1235,
       "name" : "new-server-test",
       "imageId" : 1,
       "flavorId" : 1,
       "hostId" : "e4d909c290d0fb1ca068ffaddf22cbd0",
       "progress" : 0,
       "status" : "BUILD",
       "adminPass" : "GFf1j9aP",
        "metadata" : {
            "My Server Name" : "Apache1"
        },
        "addresses" : {
            "public" : [
                "67.23.10.138"
            ],
            "private" : [
                "10.176.42.19"
            ]
        }
    }
}
```

4.1.3. Get Server Details

Verb	URI	Description
GET	/servers/*id*	List details of the specified server

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), itemNotFound (404), overLimit (413)

This operation returns the details of a specific server by its ID.

This operation does not require a request body.

Example 4.7. Server Details Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 id="1234" name="sample-server"
 imageId="1" flavorId="1"
 status="BUILD" progress="60"
        hostId="e4d909c290d0fb1ca068ffaddf22cbd0"
 >
  <metadata>
    <meta key="Server Label">Web Head 1</meta>
    <meta key="Image Version">2.1</meta>
  </metadata>
  <addresses>
    <public>
      <ip addr="67.23.10.132"/>
      <ip addr="67.23.10.131"/>
    </public>
    <private>
      <ip addr="10.176.42.16"/>
    </private>
  </addresses>
</server>
```

Example 4.8. Server Details Response: JSON

```
{
    "server" : {
        "id" : 1234,
        "name" : "sample-server",
        "imageId" : 1,
        "flavorId" : 1,
        "hostId" : "e4d909c290d0fb1ca068ffaddf22cbd0",
        "status" : "BUILD",
        "progress" : 60,
        "addresses" : {
            "public" : [
                "67.23.10.132",
                "67.23.10.131"
            ],
            "private" : [
                "10.176.42.16"
            ]
        },
        "metadata" : {
            "Server Label" : "Web Head 1",
            "Image Version" : "2.1"
        }
    }
}
```

4.1.4. Update Server Name / Administrative Password

Verb	URI	Description
PUT	/servers/*id*	Update the specified server's name and/or administrative password

Normal Response Code(s): 204

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), overLimit (413)

Status Transition: ACTIVE → PASSWORD → ACTIVE

This operation allows you to update the name of the server and/or change the administrative password. This operation changes the name of the server in the Cloud Servers system and does not change the server host name itself.

Example 4.9. Server Update Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 name="new-server-test" adminPass="newPassword" />
```

Example 4.10. Server Update Request: JSON

```
{
  "server" :
    {
        "name" : "new-server-test",
     "adminPass" : "newPassword"
    }
}
```

This operation does not contain a response body.

4.1.5. Delete Server

Verb	URI	Description
DELETE	/servers/*id*	Terminate the specified server

Normal Response Code(s): 204

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), itemNotFound (404), unauthorized (401), buildInProgress (409), overLimit (413)

Status Transition:	`ACTIVE` → `DELETED`
	`SUSPENDED` → `DELETED`

This operation deletes a cloud server instance from the system.

Note

When a server is deleted, all images created from that server are also removed.

This operation does not require a request or a response body.

4.2. Server Addresses

4.2.1. List Addresses

Verb	URI	Description
GET	/servers/*id*/ips	List all server addresses

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation does not require a request body.

Example 4.11. Addresses List Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<addresses xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <public>
    <ip addr="67.23.10.132"/>
    <ip addr="67.23.10.131"/>
  </public>
  <private>
    <ip addr="10.176.42.16"/>
  </private>
</addresses>
```

Example 4.12. Addresses List Response: JSON

```
{
    "addresses" : {
        "public" : [
            "67.23.10.132",
            "67.23.10.131"
        ],
        "private" : [
            "10.176.42.16"
        ]
    }
}
```

4.2.2. List Public Addresses

Verb	URI	Description
GET	/servers/*id*/ips/public	List all public server addresses

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation does not require a request body.

Example 4.13. Public Addresses List Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<public xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <ip addr="67.23.10.132"/>
  <ip addr="67.23.10.131"/>
</public>
```

Example 4.14. Public Addresses List Response: JSON

```
{
    "public" : [
        "67.23.10.132",
        "67.23.10.131"
    ]
}
```

4.2.3. List Private Addresses

Verb	URI	Description
GET	/servers/*id*/ips/private	List all private server addresses

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation does not require a request body.

Example 4.15. Private Addresses List Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<private xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <ip addr="10.176.42.16"/>
</private>
```

Example 4.16. Private Addresses List Response: JSON

```
{
    "private" : [
        "10.176.42.16"
    ]
}
```

4.2.4. Share an IP Address

Verb	URI	Description
PUT	/servers/*id*/ips/public/*address*	Share an IP address to the specified server

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), overLimit (413)

Status Transition:	ACTIVE → SHARE_IP → ACTIVE (if configureServer is true)
	ACTIVE → SHARE_IP_NO_CONFIG → ACTIVE

This operation shares an IP from an existing server in the specified shared IP group to another specified server in the same group. By default, the operation modifies cloud network restrictions to allow IP traffic for the given IP to/from the server specified, but does not bind the IP to the server itself. A heartbeat facility (e.g. keepalived) can then be used within the servers to perform health checks and manage IP failover. If the configureServer attribute is set to true, the server is configured with the new address, though the address is not enabled. Note that configuring the server does require a reboot.

Example 4.17. Share IP Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<shareIp xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  sharedIpGroupId="1234" configureServer="true" />
```

Example 4.18. Share IP Response: JSON

```
{
    "shareIp" : {
        "sharedIpGroupId" : 1234,
        "configureServer" : true
    }
}
```

This operation does not return a response body.

4.2.5. Unshare an IP Address

Verb	URI	Description
DELETE	/servers/*id*/ips/public/*address*	Remove a shared IP address from the specified server

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), overLimit (413)

Status Transition:	`ACTIVE` → `DELETE_IP` → `ACTIVE`

This operation removes a shared IP address from the specified server.

This operation does not contain a request or response body.

4.3. Server Actions

4.3.1. Reboot Server

Verb	URI	Description
POST	/servers/*id*/action	Reboot the specified server

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), overLimit (413)

Status Transition:	ACTIVE → REBOOT → ACTIVE (soft reboot)
	ACTIVE → HARD_REBOOT → ACTIVE (hard reboot)

The reboot function allows for either a soft or hard reboot of a server. With a soft reboot (SOFT), the operating system is signaled to restart, which allows for a graceful shutdown of all processes. A hard reboot (HARD) is the equivalent of power cycling the server.

Example 4.19. Action Reboot: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<reboot xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 type="HARD"/>
```

Example 4.20. Action Reboot: JSON

```
{
    "reboot" : {
        "type" : "HARD"
    }
}
```

This operation does not return a response body.

4.3.2. Rebuild Server

Verb	URI	Description
POST	/servers/*id*/action	Rebuild the specified server

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), serverCapacityUnavailable (503), overLimit (413)

Status Transition:	ACTIVE → REBUILD → ACTIVE
	ACTIVE → REBUILD → ERROR (on error)

The rebuild function removes all data on the server and replaces it with the specified image. serverId and IP addresses will remain the same.

Example 4.21. Action Rebuild: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<rebuild xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  imageId="2"/>
```

Example 4.22. Action Rebuild: JSON

```
{
    "rebuild" : {
        "imageId" : 2
    }
}
```

This operation does not return a response body.

4.3.3. Resize Server

Verb	URI	Description
POST	/servers/*id*/action	Resize the specified server

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), serverCapacityUnavailable (503), overLimit (413), resizeNotAllowed (403)

Status Transition:	ACTIVE → QUEUE_RESIZE → PREP_RESIZE → RESIZE → VERIFY_RESIZE
	ACTIVE → QUEUE_RESIZE → ACTIVE (on error)

The resize function converts 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 they are not explicitly confirmed or reverted.

Example 4.23. Action Resize: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<resize xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 flavorId="3"/>
```

Example 4.24. Action Resize: JSON

```
{
    "resize" : {
        "flavorId" : 3
    }
}
```

This operation does not return a response body.

4.3.4. Confirm Resized Server

Verb	URI	Description
POST	/servers/*id*/action	Confirm a pending resize action

Normal Response Code(s): 204

Error ResponseCode(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), serverCapacityUnavailable (503), overLimit (413), resizeNotAllowed (403)

Status Transition:	VERIFY_RESIZE → ACTIVE

During a resize operation, the original server is saved for a period of time to allow roll back if there is a problem. Once the newly resized server is tested and has been confirmed to be functioning properly, use this operation to confirm the resize. After confirmation, the original server is removed and cannot be rolled back to. All resizes are automatically confirmed after 24 hours if they are not explicitly confirmed or reverted.

Example 4.25. Action Confirm Resize: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<confirmResize xmlns="http://docs.rackspacecloud.com/servers/api/v1.0" />
```

Example 4.26. Action Confirm Resize: JSON

```
{
    "confirmResize" : null
}
```

This operation does not return a response body.

4.3.5. Revert Resized Server

Verb	URI	Description
POST	/servers/*id*/action	Cancel and revert a pending resize action

Normal Response Code(s): 202

Error ResponseCode(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), serverCapacityUnavailable (503), overLimit (413), resizeNotAllowed (403)

Status Transition:	VERIFY_RESIZE → ACTIVE

During a resize operation, the original server is saved for a period of time to allow for roll back if there is a problem. If you determine there is a problem with a newly resized server, use this operation to revert the resize and roll back to the original server. All resizes are automatically confirmed after 24 hours if they have not already been confirmed explicitly or reverted.

Example 4.27. Action Revert Resize: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<revertResize xmlns="http://docs.rackspacecloud.com/servers/api/v1.0" />
```

Example 4.28. Action Revert Resize: JSON

```
{
    "revertResize" : null
}
```

This operation does not return a response body.

4.4. Flavors

A flavor is an available hardware configuration for a server. Each flavor has a unique combination of disk space and memory capacity.

4.4.1. List Flavors

Verb	URI	Description
GET	/flavors	List available flavors (IDs and names only)
GET	/flavors/detail	List available flavors (all details)

Normal Response Code(s): 200, 203

Error ResponseCode(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation will list all available flavors with details.

This operation does not require a request body.

Example 4.29. Flavors List Response: XML (detail)

```
<?xml version="1.0" encoding="UTF-8"?>

<flavors  xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <flavor id="1" name="256 MB Server"
          ram="256" disk="10"  />
  <flavor id="2" name="512 MB Server"
          ram="512" disk="20" />
</flavors>
```

Example 4.30. Flavors List Response: JSON (detail)

```
{
    "flavors" : [
        {
            "id" : 1,
            "name" : "256 MB Server",
            "ram"  : 256,
            "disk" : 10
        },
        {
            "id" : 2,
            "name" : "512 MB Server",
            "ram"  : 512,
            "disk" : 20
        }
    ]
}
```

4.4.2. Get Flavor Details

Verb	URI	Description
GET	/flavors/*id*	List details of the specified flavor

Normal Response Code(s): 200, 203

Error ResponseCode(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), itemNotFound (404), overLimit (413)

This operation returns details of the specified flavor.

This operation does not require a request body.

Example 4.31. Flavor Details Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<flavor  xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  id="1" name="256 MB Server"
  ram="256" disk="10" />
```

Example 4.32. Flavor Details Response: JSON

```
{
    "flavor" : {
        "id" : 1,
        "name" : "256 MB Server",
        "ram" : 256,
        "disk" : 10
    }
}
```

4.5. Images

An image is a collection of files you use to create or rebuild a server. Rackspace provides pre-built OS images by default. You may also create custom images.

4.5.1. List Images

Verb	URI	Description
GET	/images	List available images (IDs and names only)
GET	/images/detail	List available images (all details)

Normal Response Code(s): 200, 203

Error ResponseCode(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation will list all images visible by the account.

In-flight images will have the status attribute set to `SAVING` and the conditional progress element (0-100% completion) will also be returned. Other possible values for the status attribute include: `UNKNOWN`, `PREPARING`, `ACTIVE`, `QUEUED`, `FAILED`. Images with an `ACTIVE` status are available for install.

This operation does not require a request body.

Example 4.33. Images List Response: XML (detail)

```
<?xml version="1.0" encoding="UTF-8"?>

<images xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <image id="1"   name="CentOS 5.2"
  updated="2010-10-10T12:00:00Z"
  created="2010-08-10T12:00:00Z"
  status="ACTIVE"
  />
  <image id="743" name="My Server Backup"
  serverId="12"
  updated="2010-10-10T12:00:00Z"
  created="2010-08-10T12:00:00Z"
  status="SAVING" progress="80"
  />
</images>
```

Example 4.34. Images List Response: JSON (detail)

```
{
    "images" : [
        {
            "id" : 1,
            "name" : "CentOS 5.2",
            "updated" : "2010-10-10T12:00:00Z",
            "created" : "2010-08-10T12:00:00Z",
            "status" : "ACTIVE"
        },
        {
            "id" : 743,
            "name" : "My Server Backup",
            "serverId" : 12,
            "updated" : "2010-10-10T12:00:00Z",
            "created" : "2010-08-10T12:00:00Z",
            "status" : "SAVING",
            "progress" : 80
        }
    ]
}
```

4.5.2. Create Image

Verb	URI	Description
POST	/images	Create a new image

Normal Response Code(s): 202

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badMediaType(415), itemNotFound (404), badRequest (400), serverCapacityUnavailable (503), buildInProgress (409), resizeNotAllowed (403), backupOrResizeInProgress (409), overLimit (413)

Status Transition:	`QUEUED` → `PREPARING` → `SAVING` → `ACTIVE`
	`QUEUED` → `PREPARING` → `SAVING` → `FAILED` (on error)

This operation creates a new image for the given server ID. Once complete, a new image will be available that can be used to rebuild or create servers. The image creation status can be queried by performing a **GET** on /images/*`id`* and examining the status and progress attributes.

Note

At present, image creation is an asynchronous operation, so coordinating the creation with data quiescence, etc. is currently not possible.

Example 4.35. Image Create Request: XML

```
<image xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  name="Just in case" serverId="12" />
```

Example 4.36. Image Create Request: JSON

```
{
      "image" : {
          "serverId" : 12,
          "name" : "Just in case"
      }
}
```

Example 4.37. Image Create Response: XML

```
<image xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
         id="22" name="Just in case"
         created="2010-10-10T12:00:00Z"
         status="SAVING" progress="0" serverId="12"/>
```

Example 4.38. Image Create Response: JSON

```
{
      "image" : {
          "id" : 22,
          "serverId" : 12,
          "name" : "Just in case",
          "created" : "2010-10-10T12:00:00Z",
          "status" : "SAVING",
          "progress" : 0
      }
}
```

4.5.3. Get Image Details

Verb	URI	Description
GET	/images/*id*	List details of the specified image

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), itemNotFound (404), overLimit (413)

This operation returns details of the specified image.

Note

The response body does not include the serverId field. To retrieve the serverId field, get details for all images. See Section 4.5.1, “List Images” [41]

This operation does not require a request body.

Example 4.39. Image Details Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<image xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
       id="1" name="CentOS 5.2"
       updated="2010-10-10T12:00:00Z"
       created="2010-08-10T12:00:00Z"
       status="SAVING" progress="80" />
```

Example 4.40. Image Details Response: JSON

```
{
    "image" : {
        "id" : 1,
        "name" : "CentOS 5.2",
        "updated" : "2010-10-10T12:00:00Z",
        "created" : "2010-08-10T12:00:00Z",
        "status" : "SAVING",
        "progress" : 80
    }
}
```

4.5.4. Delete Image

Verb	URI	Description
DELETE	/images/*id*	Deletes the specified image.

Normal Response Code(s): 204

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), itemNotFound (404), overLimit (413)

This operation deletes an image from the system.

Images are immediately removed. Currently, there are no state transitions to track the delete operation.

This operation does not require a request body.

This operation does not contain a response body.

4.6. Backup Schedules

In addition to creating images on demand, you may also schedule periodic (daily and weekly) images via a backup schedule. The daily and weekly images are triggered automatically based on the backup schedule established. The days/times specified for the backup schedule are targets and actual start and completion times may vary based on other activity in the system. All backup times are in GMT.

Table 4.1. Weekly Backup Schedule

Value	Day
DISABLED	Weekly backup disabled
SUNDAY	Sunday
MONDAY	Monday
TUESDAY	Tuesday
WEDNESDAY	Wednesday
THURSDAY	Thursday
FRIDAY	Friday
SATURDAY	Saturday

Table 4.2. Daily Backup Schedule

Value	Hour Range
DISABLED	Daily backups disabled
H_0000_0200	0000-0200
H_0200_0400	0200-0400
H_0400_0600	0400-0600
H_0600_0800	0600-0800
H_0800_1000	0800-1000
H_1000_1200	1000-1200
H_1200_1400	1200-1400
H_1400_1600	1400-1600
H_1600_1800	1600-1800
H_1800_2000	1800-2000
H_2000_2200	2000-2200
H_2200_0000	2200-0000

4.6.1. List Backup Schedules

Verb	URI	Description
GET	/servers/*id*/backup_schedule	List the backup schedule for the specified server

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), itemNotFound (404), overLimit (413)

This operation lists the backup schedule for the specified server.

This operation does not require a request body.

Example 4.41. Backup Schedule List Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<backupSchedule
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    enabled="true" weekly="THURSDAY" daily="H_0400_0600" />
```

Example 4.42. Backup Schedule List Response: JSON

```
{
    "backupSchedule" : {
        "enabled" : true,
        "weekly" : "THURSDAY",
        "daily" : "H_0400_0600"
    }
}
```

4.6.2. Create / Update Backup Schedule

Verb	URI	Description
POST	/servers/*id*/backup_schedule	Enable/update the backup schedule for the specified server

Normal Response Code(s): 204

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), badMediaType(415), itemNotFound (404), buildInProgress (409), serverCapacityUnavailable (503), backupOrResizeInProgress(409), overLimit (413)

This operation creates a new backup schedule or updates an existing backup schedule for the specified server. Backup schedules will occur only when the enabled attribute is set to true. The weekly and daily attributes can be used to set or to disable individual backup schedules.

Example 4.43. Backup Schedule Update Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<backupSchedule
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    enabled="true" weekly="THURSDAY" daily="H_0400_0600" />
```

Example 4.44. Backup Schedule Update Request: JSON

```
{
    "backupSchedule" : {
        "enabled" : true,
        "weekly" : "THURSDAY",
        "daily" : "H_0400_0600"
    }
}
```

This operation does not return a response body.

4.6.3. Disable Backup Schedule

Verb	URI	Description
DELETE	/servers/*id*/backup_schedule	Disables the backup schedule for the specified server

Normal Response Code(s): 204

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), itemNotFound (404), buildInProgress (409), serverCapacityUnavailable (503), backupOrResizeInProgress(409), overLimit (413)

This operation disables the backup schedule for the specified server.

This operation does not require a request body.

This operation does not return a response body.

4.7. Shared IP Groups

A shared IP group is a collection of servers that can share IPs with other members of the group. Any server in a group can share one or more public IPs with any other server in the group. With the exception of the first server in a shared IP group, servers must be launched into shared IP groups. A server may only be a member of one shared IP group.

4.7.1. List Shared IP Groups

Verb	URI	Description
GET	/shared_ip_groups	List shared ip groups (IDs and names only)
GET	/shared_ip_groups/detail	List shared ip groups (all details)

Normal Response Code(s): 200, 203

Error ResponseCode(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

This operation provides a list of shared IP groups associated with your account.

This operation does not require a request body.

Example 4.45. Shared IP Groups List Response: XML (detail)

```xml
<?xml version="1.0" encoding="UTF-8"?>

<sharedIpGroups
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <sharedIpGroup id="1234" name="Shared IP Group 1">
    <servers>
      <server id="422" />
      <server id="3445" />
    </servers>
  </sharedIpGroup>
  <sharedIpGroup id="5678" name="Shared IP Group 2">
    <servers>
      <server id="23203"/>
      <server id="2456" />
      <server id="9891" />
    </servers>
  </sharedIpGroup>
</sharedIpGroups>
```

Example 4.46. Shared IP Groups List Response: JSON (detail)

```json
{
    "sharedIpGroups" : [
        {
            "id" : 1234,
            "name" : "Shared IP Group 1",
            "servers" : [422, 3445]
        },
        {
            "id" : 5678,
            "name" : "Shared IP Group 2",
            "servers" : [23203, 2456, 9891]
        }
    ]
}
```

4.7.2. Create Shared IP Group

Verb	URI	Description
POST	/shared_ip_groups	Create a new shared ip group

Normal Response Code(s): 201

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badMediaType(415), badRequest (400), overLimit (413)

This operation creates a new shared IP group. Please note, all responses to requests for shared_ip_groups return an array of servers. However, on a create request, the shared IP group can be created empty or can be initially populated with a single server. Submitting a create request with a sharedIpGroup that contains an array of servers will generate a badRequest (400) fault.

Example 4.47. Shared IP Group Create Request: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<sharedIpGroup
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  name="Shared IP Group 1">
  <server id="422"/>
</sharedIpGroup>
```

Example 4.48. Shared IP Group Create Request: JSON

```
{
    "sharedIpGroup" : {
        "name" : "Shared IP Group 1",
        "server" : 422
    }
}
```

Example 4.49. Shared IP Group Create Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<sharedIpGroup
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    id="1234" name="Shared IP Group 1">
  <servers>
    <server id="422"/>
  </servers>
```

```
</sharedIpGroup>
```

Example 4.50. Shared IP Group Create Response: JSON

```
{
    "sharedIpGroup" : {
        "id" : 1234,
        "name" : "Shared IP Group 1",
        "servers" : [422]
    }
}
```

4.7.3. Get Shared IP Group Details

Verb	URI	Description
GET	/shared_ip_groups/*id*	List details of the specified shared IP group

Normal Response Code(s): 200, 203

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), badRequest (400), itemNotFound (404), overLimit (413)

This operation returns details of the specified shared IP group.

Example 4.51. Shared IP Group Details Response: XML

```
<?xml version="1.0" encoding="UTF-8"?>

<sharedIpGroup
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    id="1234" name="Shared IP Group 1">
  <servers>
    <server id="422"/>
  </servers>
</sharedIpGroup>
```

Example 4.52. Shared IP Group Details Response: JSON

```
{
    "sharedIpGroup" : {
        "id" : 1234,
        "name" : "Shared IP Group 1",
        "servers" : [422]
    }
}
```

This operation does not require a request body.

4.7.4. Delete Shared IP Group

Verb	URI	Description
DELETE	/shared_ip_groups/*id*	Delete the specified shared IP group

Normal Response Code(s): 204

Error Response Code(s): cloudServersFault (400, 500), serviceUnavailable (503), unauthorized (401), itemNotFound (404), overLimit (413)

This operation deletes the specified shared IP group. This operation will **only** succeed if:

1. There are no active servers in the group (i.e. they have all been terminated) or
2. No servers in the group are actively sharing IPs.

This operation does not require a request body.

This operation does not contain a response body.