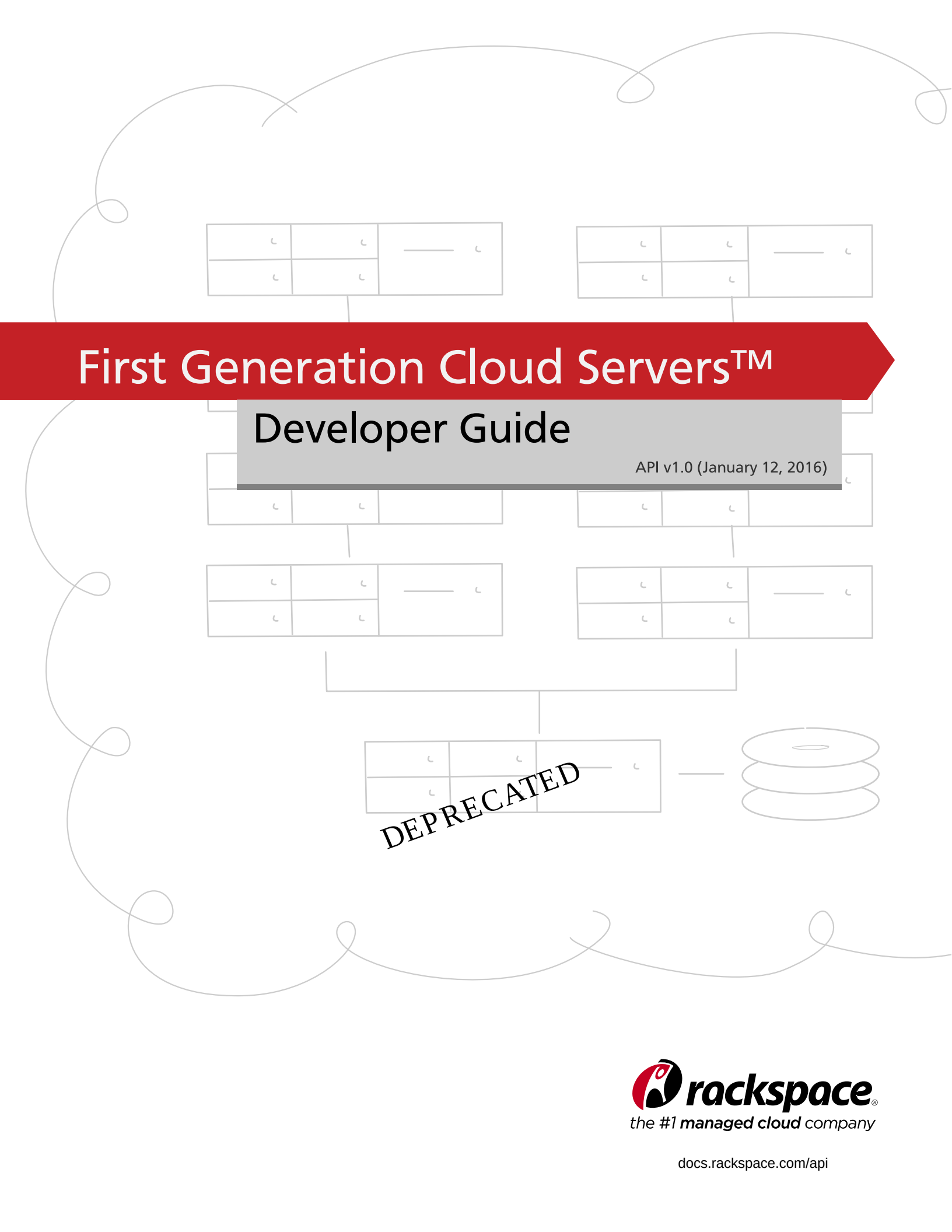

First Generation Cloud Servers™

Developer Guide

API v1.0 (January 12, 2016)

DEPRECATED

rackspace®
the #1 managed cloud company

docs.rackspace.com/api

First Generation Cloud Servers™ Developer Guide

API v1.0 (2016-01-12)

This document is intended for software developers who want to develop applications by using the first generation Rackspace Cloud Servers™ Application Programming Interface (API). The document is for informational purposes only and is provided "AS IS."

Table of Contents

List of Tables

List of Examples

Preface

The first generation 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 can 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 through the Rackspace Cloud Control Panel (GUI) or programmatically through the Rackspace Cloud Servers API.

Important

During 2015, all First Generation servers will be migrated to Next Generation servers, on a rolling basis.

Notification from Rackspace will be sent to customers informing them of their 30-day window to complete the migration of the specified servers. If you take no action, your server will be migrated for you.

To migrate your servers, during your migration window, simply perform a S0FT reboot, either from the Control Panel or by using the reboot API operation (see Section 3.3.1, "Reboot Server" [38]). The migration process will preserve all data and configuration settings.

During the 30-day migration window, performing a HARD reboot will reboot the server without triggering the migration.

You can see information about your migration window's open and close dates, use the Get Server Details operation (see Section 3.1.3, "Get Server Details" [29]), and look in the metadata section of the response for "FG2NG_self_migration_available_till" and "FG2NG_self_migration_available_from" key pairs. If your migration window has not been scheduled, you will not see these metadata keys.

For more information about the server migration see the Knowledge Center article: *First Generation to Next Generation cloud server migration FAQ*

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

1. Intended Audience

This guide is intended to assist software developers who want to develop applications by using the first generation Rackspace Cloud Servers API.

To use this information, you should have access to an active Rackspace Cloud Servers account, and you should also be familiar with the following concepts:

- Rackspace Cloud Servers service
- ReSTful web services
- HTTP/1.1

- JSON and/or XML data serialization formats

2. Document Change History

This version of the document replaces and obsoletes all previous versions. The following table describes the most recent changes:

Revision Date	Summary of Changes
January 30, 2015	• Updated following sections with information about the migration of First Gen servers to Next Gen servers: Preface [viii], Section 3.3.1, "Reboot Server" [38], and Section 3.1.3, "Get Server Details" [29].
January 17, 2014	• Updated Section 3.1.4, "Update Server Name / Administrative Password" [31]
November 4, 2013	• Added warning about 1st gen servers and new customers: Section 4, "Pricing and Service Level" [xii] and Section 3.1.2, "Create Server" [25]
September 26, 2013	• Added section Section 2.12, "Role Based Access Control" [20]
June 24, 2013	• Added important note that user may choose any endpoint regardless of account origin.
May 23, 2013	• Updated "control panel" to "Control Panel."
January 10, 2013	• Updated Section 3.1.2, "Create Server" [25] to change file size limit for file injection from 10 KB to 2 KB.
October 16, 2012	• Updated the list servers API operation to include cURL examples of the list servers request.
September 21, 2012	• Updated the create backup schedule API operation to list 202 Accepted as the successful response code. See Section 3.6.2, "Create / Update Backup Schedule" [57].
September 17, 2012	• Added pricing and service level agreement. See Section 4, "Pricing and Service Level" [xii].
September 7, 2012	• Updated the list images JSON and XML examples to reflect the latest images. • Updated the shared IP group feature description in the Section 3.1.2, "Create Server" [25].
August 31, 2012	• Updated JSON request example in resize server to remove quotes from integer value. See Example 3.25, "Resize Server: JSON Request" [41].
August 15, 2012	• First generation Cloud Servers UK launch. Added authentication endpoint for UK accounts.
August 7, 2012	• Updated the shared IP group feature description in the Section 3.1.2, "Create Server" [25].
August 1, 2012	• Updated examples for list servers, create server, get server details, update server name, list images, get image details, list flavors, get flavor details, resize server, and rebuild server operations. • Moved JSON examples before the XML examples because JSON is the default. • Added a note to the create shared IP group operation.
July 18, 2012	• Updated examples in versions calls.
July 9, 2012	• Changes to authentication and the Cloud services catalog.
June 21, 2012	• Reduced size of title in the PDF.
June 7, 2012	• Updated title to First Generation Cloud Servers.
May 22, 2012	• Added workaround information to the "Caching" section for the issue with the request header being ignored by the varnish server. • Updated get server details with information about fields returned in the response body. • Updated list servers with new response examples.
May 17, 2012	• Updated the share an IP address and unshare an IP address operations with details about URI parameters and request body attributes. • Updated the get image operation to correct the XML and JSON examples. • Updated descriptions for API operations to make them consistent.
May 16, 2012	• Updated error on the share an IP address operation page.

Revision Date	Summary of Changes
	• Minor update for get image details.
May 8, 2012	• Updated Section 3.1.2, "Create Server" [25] with guidelines about file injection. • Updated Section 3.1.2, "Create Server" [25] with details about using shared IP addresses for high availability.
April 19, 2012	• Updated references for "contact your cloud provider" to "contact Rackspace support."
April 10, 2012	• Added link to *Cloud Authentication Client Developer Guide* in the Authentication section. • Added descriptions of server statuses to the list servers API operation.
March 20, 2012	• Added note to get image details about missing serverId field in the response body.
September 14, 2011	• Added cURL example in the Authentication section.
March 10, 2011	• Fixed error referring to X-Auth-User instead of X-Auth-Key.
January 12, 2011	• Added section numbers.
January 4, 2011	• Expanded authentication information for UK release.
October 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.
September 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.
August 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.
July 14, 2009	• Initial release.

3. Resources

Use the following resources in conjunction with this guide:

- For the latest version of this guide, see *First Generation Rackspace Cloud Servers Developer Guide* on the Rackspace Cloud website. This site also provides the following related documents: *First Generation Cloud Servers Release Notes*, *Cloud Servers API Schema Types* and *Cloud Servers Frequently Asked Questions*.

- For details about authentication, see *Cloud Identity Client Developer Guide v1.1* and *Cloud Identity Client Developer Guide v2.0*.

- For details about the Cloud Servers service, see the http://www.rackspacecloud.com/cloud_hosting_products/servers site. This site also provides related documents and links to official support channels for Rackspace, including knowledge base articles, phone, chat, and email.

- To follow updates and announcements at through twitter, see http://twitter.com/rackspace.

4. Pricing and Service Level

First generation Cloud Servers is part of the Rackspace Cloud and your use through the API will be billed as per the pricing schedule at http://www.rackspace.com/cloud/public/servers/pricing.

The Service Level Agreement (SLA) for Cloud Servers is available at http://www.rackspace.com/cloud/legal/sla/#cloud_servers.

Note

Customers who sign up for Rackspace Cloud Servers after October 10, 2013, can create only Next Generation servers. To learn more, see http://docs.rackspace.com/servers/api/v2/cs-devguide/content/CreateServers.html..

1. Concepts

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

1.1. Server

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

1.2. 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.

1.3. 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.

1.4. Backup Schedule

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

1.5. 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.

1.6. 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.

1.7. 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.

1.8. 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 (such as, keepalived) that monitor for failure and manage IP failover.

1.9. 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 be a member of only one shared IP group.

2. 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.

2.1. Authentication through the Rackspace Cloud Identity Service

To authenticate, you issue an authentication request to the Rackspace Cloud Identity Service, which is an implementation of the OpenStack Keystone Identity Service v2.0.

Important

Multiple Rackspace Cloud Identity Service endpoints exist. You may use any endpoint, regardless of where your account was created.

When you authenticate, use the chosen endpoint, as follows:

National location	Rackspace Cloud Identity Service endpoint
US	https://identity.api.rackspacecloud.com/v2.0
UK	https://lon.identity.api.rackspacecloud.com/v2.0

In response to valid credentials, an authentication request to the Rackspace Cloud Identity Service returns an authentication token and a service catalog that contains a list of all services and endpoints available for this token. Because the authentication token expires after 24 hours, you must generate a new token once a day.

The following sections show you how to embed an authentication request in a cURL command to call the Rackspace Cloud Identity Service to get an authentication token and a service catalog.

For detailed information about the OpenStack Keystone Identity Service v2.0, see *Cloud Identity Client Developer Guide API v2.0*. For information about support for legacy identity endpoints, see Alternate Authentication Endpoints.

2.1.1. Authentication Request

To authenticate, you issue a **POST /tokens** request to the appropriate Rackspace Cloud Identity Service endpoint.

In the request body, supply one of the following:

- username and password
- username and API key

Your username and password are the ones that you use to log into the Rackspace Cloud Control Panel.

To obtain your API key, log into http://mycloud.rackspace.com, click your username, and select **API Keys** to get your key.

The following cURL examples show how to get an authentication token by entering either your username and password, or username and API key.

Example 2.1. Authenticate to US Identity Endpoint – Username and Password: JSON Request

```
curl -s https://identity.api.rackspacecloud.com/v2.0/tokens -X 'POST' \
     -d '{"auth":{"passwordCredentials":{"username":"theUser", "password":"thePassword"}}}' \
     -H "Content-Type: application/json" | python -m json.tool
```

Example 2.2. Authenticate to US Identity Endpoint – Username and API Key: JSON Request

```
curl -s https://identity.api.rackspacecloud.com/v2.0/tokens -X 'POST' \
     -d '{"auth":{"RAX-KSKEY:apiKeyCredentials":{"username":"theUserName", "apiKey":"theAPIKey"}}}' \
     -H "Content-Type: application/json" | python -m json.tool
```

Note

In these examples, the following code is appended to the cURL commands to pretty-print the JSON output:

Example 2.3. Pretty Printing cURL Output

```
| python -m json.tool
```

2.1.2. Authentication Response

In response to valid credentials, your request returns an authentication token and a service catalog with the endpoints to request services.

The following output shows a sample authentication response in JSON format:

Example 2.4. Authenticate: JSON Response

```
{
    "access": {
        "serviceCatalog": [
            {
                "endpoints": [
                    {
                        "publicURL": "https://dfw.loadbalancers.api.rackspacecloud.com/v1.0/345789",
                        "region": "DFW",
                        "tenantId": "345789"
                    },
                    {
                        "publicURL": "https://ord.loadbalancers.api.rackspacecloud.com/v1.0/345789",
                        "region": "ORD",
                        "tenantId": "345789"
                    }
                ],
                "name": "cloudLoadBalancers",
                "type": "rax:load-balancer"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://servers.api.rackspacecloud.com/v1.0/345789",
                        "tenantId": "345789",
                        "versionId": "1.0",
                        "versionInfo": "https://servers.api.rackspacecloud.com/v1.0",
                        "versionList": "https://servers.api.rackspacecloud.com/"
                    }
                ],
                "name": "cloudServers",
                "type": "compute"
            },
```

```
            {
                "endpoints": [
                    {
                        "publicURL": "https://dfw.databases.api.rackspacecloud.com/v1.0/345789",
                        "region": "DFW",
                        "tenantId": "345789"
                    },
                    {
                        "publicURL": "https://ord.databases.api.rackspacecloud.com/v1.0/345789",
                        "region": "ORD",
                        "tenantId": "345789"
                    }
                ],
                "name": "cloudDatabases",
                "type": "rax:database"
            },
            {
                "endpoints": [
                    {
                        "internalURL": "https://snet-storage101.dfw1.clouddrive.com/v1/
MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7",
                        "publicURL": "https://storage101.dfw1.clouddrive.com/v1/
MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7",
                        "region": "DFW",
                        "tenantId": "MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7"
                    }
                ],
                "name": "cloudFiles",
                "type": "object-store"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://preprod.dfw.servers.api.rackspacecloud.com/v2/345789",
                        "region": "DFW",
                        "tenantId": "345789"
                    }
                ],
                "name": "cloudServersPreprod",
                "type": "compute"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://dns.api.rackspacecloud.com/v1.0/345789",
                        "tenantId": "345789"
                    }
                ],
                "name": "cloudDNS",
                "type": "rax:dns"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://ord.servers.api.rackspacecloud.com/v2/345789",
                        "region": "ORD",
                        "tenantId": "345789",
                        "versionId": "2",
                        "versionInfo": "https://ord.servers.api.rackspacecloud.com/v2",
                        "versionList": "https://ord.servers.api.rackspacecloud.com/"
                    },
                    {
                        "publicURL": "https://dfw.servers.api.rackspacecloud.com/v2/345789",
                        "region": "DFW",
                        "tenantId": "345789",
                        "versionId": "2",
                        "versionInfo": "https://dfw.servers.api.rackspacecloud.com/v2",
                        "versionList": "https://dfw.servers.api.rackspacecloud.com/"
                    }
                ],
                "name": "cloudServersOpenStack",
                "type": "compute"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://monitoring.api.rackspacecloud.com/v1.0/345789",
                        "tenantId": "345789"
                    }
```

```
                ],
                "name": "cloudMonitoring",
                "type": "rax:monitor"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://cdn1.clouddrive.com/v1/MossoCloudFS_530f8649-324c-499c-
a075-2195854d52a7",
                        "region": "DFW",
                        "tenantId": "MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7"
                    }
                ],
                "name": "cloudFilesCDN",
                "type": "rax:object-cdn"
            }
        ],
        "token": {
            "expires": "2012-08-04T10:36:05.000-05:00",
            "id": "00d5c4bb-9451-4bf0-834a-45d427330b51",
            "tenant": {
                "id": "345789",
                "name": "345789"
            }
        },
        "user": {
            "id": "170454",
            "name": "user1234",
            "roles": [
                {
                    "description": "User Admin Role.",
                    "id": "3",
                    "name": "identity:user-admin"
                }
            ]
        }
    }
}
```


Specifically, successful authentication returns the following response information:

- **Endpoints to request Rackspace Cloud services**

  Appears in the `endpoints` field in the `serviceCatalog` element.

  The service catalog lists the public URL, which is the endpoint that you use to access the service, the region, service name, tenant ID, the version ID, and endpoints that you can use to get version information for the API.

  Do not include explicit API endpoints in your scripts and applications. Instead, find the endpoint for your service and region.

- **The name of the service**

  Appears in the `name` field in the `serviceCatalog` element.

  Locate the correct Cloud Servers endpoints in the service catalog by examining the service `name` attribute, as follows:

Cloud Servers generation	**Service name in the catalog**
First generation	`cloudServers` If you use the authentication token to access this service, you can view and perform first generation Cloud Servers API operations against your first generation Cloud Servers.
Next generation	`cloudServersOpenStack`

Cloud Servers generation	Service name in the catalog
	Next generation Cloud Servers might show multiple endpoints to enable regional choice. Select the appropriate endpoint for the region that you want to interact with by examining the `region` field. If you use the authentication token to access this service, you can view and perform next generation Cloud Servers API operations against your next generation Cloud Servers.

- **Expiration date and time for authentication token**

  Appears in the `expires` field in the `token` element.

  After this date and time, the token is no longer valid. A token may be manually revoked before the time identified by the `expires` field; this field predicts the maximum lifespan for a token, but does not guarantee that the token will reach that lifespan. Clients are encouraged to cache a token until it expires.

  Because the authentication token expires after 24 hours, you must generate a new token once a day.

- **Authentication token**

  Appears in the `id` field in the `token` element.

  You pass the authentication token in the `X-Auth-Token` header each time that you send a request to a service.

2.2. 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 2.1. JSON and XML Response Formats

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

Example 2.5. 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 2.6. 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 through 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 2.7. 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"}
}
```

2.3. Content Compression

Request and response body data may be encoded with gzip compression 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 2.2. Encoding Headers

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

2.4. Chunked Transfer Encoding

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

2.5. Persistent Connections

By default, the API supports persistent connections through 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.

2.6. 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 (such as, ?*`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.

2.7. 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 2.3. Last Modified Header

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

Note

A known problem exists with the varnish server. It does not respect the `"Cache-Control: no-cache"` header. To workaround this issue, include an unused parameter with a unique value each time you make a request. This forces the varnish server to treat the request as a new request and to bypass the cache. For example, you might add a parameter to the request like `"?dummyreq=1323710764"`, where the value is `` `date "+%s"` ``, which is the number of seconds since the epoch. Each time you make a new request, change this value. For example `"?dummyreq=1323710769"`, which forces the varnish server to bypass the cache.

2.8. 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.

2.9. 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.

2.9.1. 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 2.4. 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 an 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.

2.9.2. 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 2.5. 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

2.9.3. 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 Codes: 200 and 203

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

This operation does not require a request body.

Example 2.8. Limit: JSON Response

```
{
    "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
        }
    }
}
```

Example 2.9. Limit: XML Response

```
<?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>
```

2.10. API Version

The Cloud Servers API uses a URI versioning scheme. The first element of the path contains the target version identifier (such as, 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 2.10. Versions List Request

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

Normal Response Codes: 200 and 203

Error Response Codes: 400, 413, 500, 503

This operation does not require a request body.

Example 2.11. List Versions: JSON Response

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

Example 2.12. List Versions: XML Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<versions xmlns="http://docs.rackspacecloud.com/servers/api">
    <version status="BETA" id="v1.0"/>
</versions>
```

You can also obtain additional information about a specific version by performing a **GET** on the base version URL (such as, 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 (such as, 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 2.13. Version Details Request

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

Normal Response Codes: 200 and 203

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

This operation does not require a request body

Example 2.14. Get Version Details: JSON Response

```
{
    "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"
    }
}
```

Example 2.15. Get Version Details: XML Response

```
<?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"/>
```

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

2.11. 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 2.16. Fault: JSON Response

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

Example 2.17. Fault: XML Response

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

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 (such as, cloudServersFault) may change depending on the type of error. The following is a list of possible elements along with their associated error codes.

Table 2.6. 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 2.18. Fault, Item Not Found: JSON Response

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

Example 2.19. Fault, Item Not Found: XML Response

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

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 2.20. Fault, Over Limit: JSON Response


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


Example 2.21. Fault, Over Limit: XML Response


```
<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>
```


2.12. Role Based Access Control

Role Based Access Control (RBAC) restricts access to the capabilities of Rackspace Cloud services, including the first gen Cloud Servers API, to authorized users only. RBAC enables Rackspace Cloud customers to specify which account users of their Cloud account have access to which first gen Cloud Servers API service capabilities, based on roles defined by Rackspace (see Table 2.7, " first gen Cloud Servers Product Roles and Permissions" [21]). The permissions to perform certain operations in first gen Cloud Servers API – create, read, update, delete – are assigned to specific roles. The account owner user assigns these roles, either multiproduct (global) or product-specific (for example, first gen Cloud Servers), to account users.

2.12.1. Assigning Roles to Account Users

The account owner (identity:user-admin) can create account users on the account and then assign roles to those users. The roles grant the account users specific permissions for accessing the capabilities of the first gen Cloud Servers service. Each account has only one account owner, and that role is assigned by default to any Rackspace Cloud account when the account is created.

See the Cloud Identity Client Developer Guide for information about how to perform the following tasks:

- Create account users
- Assign roles to account users
- Delete roles from account users

Note

The account owner (identity:user-admin) role cannot hold any additional roles because it already has full access to all capabilities.

2.12.2. Roles Available for first gen Cloud Servers

Two roles (observer and admin) can be used to access the first gen Cloud Servers API specifically. The following table describes these roles and their permissions.

Table 2.7. first gen Cloud Servers Product Roles and Permissions

Role Name	Role Permissions
legacyCompute:admin	This role provides Create, Read, Update, and Delete permissions in first gen Cloud Servers, where access is granted.
legacyCompute:observer	This role provides Read permission in first gen Cloud Servers, where access is granted.

Additionally, two multiproduct roles apply to all products. Users with multiproduct roles inherit access to future products when those products become RBAC-enabled. The following table describes these roles and their permissions.

Table 2.8. Multiproduct (Global) Roles and Permissions

Role Name	Role Permissions
admin	This role provides Create, Read, Update, and Delete permissions in all products, where access is granted.
observer	This role provides Read permission in all products, where access is granted.

2.12.3. Resolving Conflicts Between RBAC Multiproduct vs. Custom (Product-specific) Roles

The account owner can set roles for both multiproduct and first gen Cloud Servers scope, and it is important to understand how any potential conflicts among these roles are resolved. When two roles appear to conflict, the role that provides the more extensive permissions takes precedence. Therefore, admin roles take precedence over observer roles, because admin roles provide more permissions.

The following table shows two examples of how potential conflicts between user roles in the Control Panel are resolved:

Permission Configuration	View of Permission in the Control Panel	Can the User Perform Product Admin Functions in the Control Panel?
User is assigned the following roles: multiproduct **observer** and first gen Cloud Servers **admin**	Appears that the user has only the multiproduct **observer** role	Yes, for first gen Cloud Servers only. The user has the **observer** role for the rest of the products.
User is assigned the following roles: multiproduct **admin** and first gen Cloud Servers **observer**	Appears that the user has only the multiproduct **admin** role	Yes, for all of the products. The first gen Cloud Servers **observer** role is ignored.

2.12.4. RBAC Permissions Cross-reference to first gen Cloud Servers API Operations

API operations for first gen Cloud Servers may or may not be available to all roles. To see which operations are permitted to invoke which calls, please review the Knowledge Center article.

3. API Operations

3.1. Servers

3.1.1. List Servers

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

Normal Response Codes: 200 and 203

Error Response Codes: 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 indicate the current server state. The server status is one of the following values:

Server Status Values

- `ACTIVE`. The server is active.
- `BUILD`. The server has not finished the original build process.
- `DELETE_IP`. An IP address is being removed from the server.
- `DELETED`. The server is deleted.
- `ERROR`. The server is in error.
- `HARD_REBOOT`. The server is hard rebooting. This is equivalent to pulling the power plug on a physical server, plugging it back in, and rebooting it.
- `PASSWORD`. The password is being reset on the server.
- `PREP_RESIZE`. The server is performing the initial copy of data to its new host. Server should still be active at this time.
- `QUEUE_RESIZE`. The server is in the queue pending a resize or migration. Server should still be active at this time.
- `REBOOT`. The server is in a soft reboot state. A reboot command was passed to the operating system.
- `REBUILD`. The server is currently being rebuilt from an image.
- `RESCUE`. The server is in rescue mode.

- `RESIZE`. Server is performing the differential copy of data that changed during its initial copy. Server is down for this stage.
- `REVERT_RESIZE`. The resize or migration of a server failed for some reason. The destination server is being cleaned up and the original source server is restarting.
- `SHARE_IP`. An IP address is currently being shared with or from this server. The network configuration for this server will change, perhaps requiring an IP address.
- `SHARE_IP_NO_CONFIG`. The host is being reconfigured to allow your server to use an IP from another machine. No configuration change should occur on your guest server, however it might reboot if the kernel is not compatible with the pause state that the hypervisor requests during the process.
- `SUSPENDED`. The server is suspended, either by request or for necessity. Review support tickets or contact Rackspace support to determine why the server is in this state.
- `UNKNOWN`. The state of the server is unknown. Contact Rackspace support.
- `VERIFY_RESIZE`. System is awaiting confirmation that the server is operational after a move or resize.

When retrieving a list of servers through the *`changes-since`* parameter (see Section 2.8, "Efficient Polling with the *`Changes-Since`* Parameter" [11].), 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 is relevant to your application.

Note

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

This operation does not require a request body.

To issue a list servers request that returns a JSON response, you can embed the request in a cURL command, as follows:

Example 3.1. List Servers: JSON Request in a cURL Command

```
$ curl -s https://servers.api.rackspacecloud.com/v1.0/$account/servers/detail
 \
     -H "X-Auth-Token: $token" | python -m json.tool
```

Where *`account`* is your tenant ID and *`token`* is your authentication token.

Example 3.2. List Servers: JSON Response (detail)

```
{
    "servers": [
        {
            "addresses": {
```

```
                "private": [
                    "10.178.54.11"
                ],
                "public": [
                    "198.101.228.60"
                ]
            },
            "flavorId": 2,
            "hostId": "2d66db781ce490432f55ada17610173e",
            "id": 21034274,
            "imageId": 119,
            "metadata": {
                "My Server Name": "API Test Server"
            },
            "name": "api-test-server",
            "progress": 100,
            "status": "ACTIVE"
        },
        {
            "addresses": {
                "private": [
                    "10.178.54.13"
                ],
                "public": [
                    "198.101.228.62"
                ]
            },
            "flavorId": 2,
            "hostId": "312452f3d7a72d3def18a06e09ec01b7",
            "id": 21034280,
            "imageId": 119,
            "metadata": {
                "My Server Name": "API Test Server XML"
            },
            "name": "api-test-server-xml",
            "progress": 100,
            "status": "ACTIVE"
        }
    ]
}
```


To issue a list servers request that returns an XML response, you can embed the request in a cURL command, as follows:

Example 3.3. List Servers: XML Request in a cURL Command

```
$ curl -i https://servers.api.rackspacecloud.com/v1.0/$account/servers/detail.
xml \
     -H "Content-Type: application/xml" \
     -H "Accept: application/xml" \
     -H "X-Auth-Token: $token"
```

Where `account` is your tenant ID and `token` is your authentication token.

Example 3.4. List Servers: XML Response (detail)


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<servers xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <server status="ACTIVE" progress="100"
```

```
    hostId="2d66db781ce490432f55ada17610173e" flavorId="2"
    imageId="119" id="21034274" name="api-test-server">
    <metadata>
      <meta key="My Server Name">API Test Server</meta>
    </metadata>
    <addresses>
      <public>
        <ip addr="198.101.228.60"/>
      </public>
      <private>
        <ip addr="10.178.54.11"/>
      </private>
    </addresses>
  </server>
  <server status="ACTIVE" progress="100"
    hostId="312452f3d7a72d3def18a06e09ec01b7" flavorId="2"
    imageId="119" id="21034280" name="api-test-server-xml">
    <metadata>
      <meta key="My Server Name">API Test Server XML</meta>
    </metadata>
    <addresses>
      <public>
        <ip addr="198.101.228.62"/>
      </public>
      <private>
        <ip addr="10.178.54.13"/>
      </private>
    </addresses>
  </server>
</servers>
```

3.1.2. Create Server

Verb	URI	Description
POST	/servers	Creates a server.

Normal Response Code: 202

Error Response Codes: 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)

Note

Customers who sign up for Rackspace Cloud Servers after October 10, 2013, can create only Next Generation servers. To learn more, see http://docs.rackspace.com/servers/api/v2/cs-devguide/content/CreateServers.html..

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. To check the progress of the request, issue a **GET** on **/server/*id*** , which returns a progress field, which represents the percentage completion and is a value from 0 to 100.

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

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

Follow these guidelines when you inject files:

- The maximum size of the file path data is 255 bytes.
- Encode the file contents as a Base64 string. The maximum size of the file contents is 2 KB (2048 bytes).

> **Note**
>
> The 2 KB limit refers to the number of bytes in the decoded data and not the number of characters in the encoded data.

- You can inject text files only. You cannot inject binary or zip files into a new build.
- You can specify a maximum of five file path/content pairs.

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

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

After file injection, all files have root and the root group as the owner and group owner, respectively, and 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 use the shared IP feature, you must plan ahead. When you create a shared IP group, you can add only one existing server to it. Additional servers can be added to it only by specifically creating these servers in that shared IP group. This limitation is a technology limitation; IP group sharing depends on the servers being located on specific hypervisors and so you cannot assign random servers to the group. After you create a shared IP group, you can add more servers to it when you create each server. You cannot add them to the group retroactively.

Procedure 3.1. To share public IPs:

1. Create a shared IP group.
2. Add servers to that group.

3. Configure the instance to use the additional IPs.

4. Configure the host firewall to allow the instance to pass traffic using the additional IPs.

5. To launch an HA server, include the optional `sharedIpGroupId` attribute. The server is launched into that shared IP group.

If you intend to use a shared IP on the server being created and you do not need a separate public IP address, you can 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 you specify the `sharedIpGroupId` attribute.

Note

The `sharedIpGroupId` attribute is optional. For optimal performance, use this attribute only when you intend to share IPs between servers.

The following table describes the fields in the response body:

Table 3.1. Create Server Response Fields

Name	Description
addresses	The private and public IP addresses for the server.
adminPass	A randomly-generated password. For security reasons, the password is not returned in subsequent **GET** calls against the server ID.
flavorId	The flavor ID.
hostId	The host ID.
id	The server ID
imageId	The image ID.
metadata	Metadata key and value pairs, if any.
name	The name of the server.
progress	The percentage value of the build status. For example, if the status is 60, the server is 60% built. This value is from 0 to 100.
status	The server status. The server status. See Server Status Values [22].

Example 3.5. Create Server: JSON Request


```
{
    "server" : {
        "name" : "api-test-server",
        "imageId" : 119,
        "flavorId" : 2,
        "metadata" : {
            "My Server Name" : "API Test Server"
        },
        "personality" : [
            {
                "path" : "/etc/banner.txt",
                "contents" : "ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBp
 dCBtb3ZlcyBpbiBqdXN0IHN1Y2ggYSBkaXJlY3Rpb24gYW5k
 IGF0IHN1Y2ggYSBzcGVlZC4uLkl0IGZlZWxzIGFuIGltcHVs
 c2lvbi4uLnRoaXMgaXMgdGhlIHBsYWNlIHRvIGdvIG5vdy4g
 QnV0IHRoZSBza3kga25vd3MgdGhlIHJlYXNvbnMgYW5kIHRo
```

```
ZSBwYXR0ZXJucyBiZWhpbmQgYWxsIGNsb3VkcywgYW5kIHlv
dSB3aWxsIGtub3csIHRvbywgd2hlbiB5b3UgbGlmdCB5b3Vy
c2VsZiBoaWdoIGVub3VnaCB0byBzZWUgYmV5b25kIGhvcml6
b25zLiINCg0KLVJpY2hhcmQgQmFjaA=="
            }
        ]
    }
}
```


Example 3.6. Create Server: JSON Response


```
{
    "server": {
        "addresses": {
            "private": [
                "10.178.54.11"
            ],
            "public": [
                "198.101.228.60"
            ]
        },
        "adminPass": "NA7io0E1lapi-test-server",
        "flavorId": 2,
        "hostId": "2d66db781ce490432f55ada17610173e",
        "id": 21034274,
        "imageId": 119,
        "metadata": {
            "My Server Name": "API Test Server"
        },
        "name": "api-test-server",
        "progress": 0,
        "status": "BUILD"
    }
}
```


Example 3.7. Create Server: XML Request


```
<?xml version="1.0" encoding="UTF-8"?>
<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
       name="api-test-server-xml"
       imageId="119"
       flavorId="2">
  <metadata>
    <meta
      key="My Server Name">API Test Server XML</meta>
  </metadata>
  <personality>
    <file
      path="/etc/banner.txt">
        ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBp
        dCBtb3ZlcyBpbiBqdXN0IHN1Y2ggYSBkaXJlY3Rpb24gYW5k
        IGF0IHN1Y2ggYSBzcGVlZC4uLkl0IGZlZWxzIGFuIGltcHVs
        c2lvbi4uLnRoaXMgaXMgdGhlIHBsYWNlIHRvIGdvIG5vdy4g
        QnV0IHRoZSBza3kga25vd3MgdGhlIHJlYXNvbnMgYW5kIHRo
        ZSBwYXR0ZXJucyBiZWhpbmQgYWxsIGNsb3VkcywgYW5kIHlv
        dSB3aWxsIGtub3csIHRvbywgd2hlbiB5b3UgbGlmdCB5b3Vy
        c2VsZiBoaWdoIGVub3VnaCB0byBzZWUgYmV5b25kIGhvcml6
        b25zLiINCg0KLVJpY2hhcmQgQmFjaA==
    </file>
```

```
  </personality>
</server>
```

Example 3.8. Create Server: XML Response


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  status="BUILD" progress="0"
  hostId="312452f3d7a72d3def18a06e09ec01b7" flavorId="2" imageId="119"
  adminPass="Whul7KK67api-test-server-xml" id="21034280"
  name="api-test-server-xml">
  <metadata>
    <meta key="My Server Name">API Test Server XML</meta>
  </metadata>
  <addresses>
    <public>
      <ip addr="198.101.228.62"/>
    </public>
    <private>
      <ip addr="10.178.54.13"/>
    </private>
  </addresses>
</server>
```


3.1.3. Get Server Details

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

Normal Response Codes: 200 and 203

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

This operation returns the details of a specified server. Specify the server ID as *id* in the URI.

This operation does not require a request body.

The response body includes the following information:

Table 3.2. Get Server Details Response Fields

Name	Description
id	The server ID.
name	The server name.
flavorId	The flavor ID.
imageId	The image ID.
status	The server status. See Server Status Values [22].
progress	The percentage value of the build status. For example, if the status is 60, the server is 60% built.
hostId	The host ID.
metadata	Metadata key and value pairs. During the 2015 migration of first gen servers to next gen servers, this section will contain key pairs indicating the start and end

Name	Description
	date of the server's migration window. See Preface [viii] for more details about the migration process.
addresses	Public and private IP addresses.

Example 3.9. Get Server Details: JSON Response

```
{
    "server": {
        "addresses": {
            "private": [
                "10.178.54.11"
            ],
            "public": [
                "198.101.228.60"
            ]
        },
        "flavorId": 2,
        "hostId": "2d66db781ce490432f55ada17610173e",
        "id": 21034274,
        "imageId": 119,
        "metadata": {
            "My Server Name": "API Test Server",
            "FG2NG_self_migration_available_till":"2015-01-30 00:00:00",
            "FG2NG_self_migration_available_from":"2014-12-14 00:00:00"
        },
        "name": "api-test-server",
        "progress": 100,
        "status": "ACTIVE"
    }
}
```

Example 3.10. Get Server Details: XML Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
  status="ACTIVE" progress="100"
  hostId="2d66db781ce490432f55ada17610173e" flavorId="2" imageId="119"
  id="21034274" name="api-test-server">
  <metadata>
    <meta key="My Server Name">API Test Server</meta>
    <meta key="FG2NG_self_migration_available_till">2015-01-30 00:00:00</meta>
    <meta key="FG2NG_self_migration_available_from">2014-12-14 00:00:00</meta>
  </metadata>
  <addresses>
    <public>
      <ip addr="198.101.228.60"/>
    </public>
    <private>
      <ip addr="10.178.54.11"/>
    </private>
  </addresses>
</server>
```

3.1.4. Update Server Name / Administrative Password

Verb	URI	Description
PUT	/servers/*id*	Updates the name and/or administrative password for the specified server.

Normal Response Code: 204

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

Status Transition:	ACTIVE → PASSWORD → ACTIVE

This operation enables you to update the name and/or change the administrative password for the specified server. This operation changes the name but does not update the host name for the server in the Cloud Servers system and does not change host name of the server. If you want to update only the server name, do not include the password in the body of the request and vice versa.

This operation does not return a response body.

Example 3.11. Update Server Name: JSON Request

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

Example 3.12. Update Server Name: XML Request

```
<?xml version="1.0" encoding="UTF-8"?>
<server xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    name="new-api-server-test-xml" adminPass="newPassword"/>
```

3.1.5. Delete Server

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

Normal Response Code: 204

Error Response Codes: 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.

3.2. Server Addresses

3.2.1. List Addresses

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

Normal Response Codes: 200 and 203

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

This operation does not require a request body.

Example 3.13. List Addresses: JSON Response


```
{
    "addresses": {
        "private": [
            "10.178.54.11"
        ],
        "public": [
            "198.101.228.60"
        ]
    }
}
```


Example 3.14. List Addresses: XML Response


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<addresses xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
  <public>
    <ip addr="198.101.228.60"/>
  </public>
  <private>
    <ip addr="10.178.54.11"/>
  </private>
</addresses>
```


3.2.2. List Public Addresses

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

Normal Response Codes: 200 and 203

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

This operation does not require a request body.

Example 3.15. List Public Addresses List: JSON Response

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

Example 3.16. List Public Addresses List: XML Response

```
<?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>
```

3.2.3. List Private Addresses

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

Normal Response Codes: 200 and 203

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

This operation does not require a request body.

Example 3.17. List Private Addresses List: JSON Response

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

Example 3.18. List Private Addresses List: XML Response

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

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

3.2.4. Share an IP Address

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

Normal Response Code: 202

Error Response Codes: 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 a specified IP address from an existing server in the specified shared IP group to another specified server in the same group.

Specify the following parameters in the URI:

id The server ID.

address The IP address that you want to share.

Specify the ID for the shared IP group in the `sharedIpGroupId` attribute in the request body.

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 (such as, 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

Configuring the server does require a reboot.

This operation requires a request body.

This operation does not return a response body.

Example 3.19. Share IP: JSON Request

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

Example 3.20. Share IP: XML Request

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

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

3.2.5. Unshare an IP Address

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

Normal Response Code: 202

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

Status Transition:	ACTIVE → DELETE_IP → ACTIVE

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

Specify the following parameters in the URI:

id The server ID.

address The shared IP address.

This operation does not contain a request body or return a response body.

3.3. Server Actions

3.3.1. Reboot Server

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

Normal Response Code: 202

Error Response Codes: 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.

Important

During 2015, all First Generation servers will be migrated to Next Generation servers, on a rolling basis.

Notification from Rackspace will be sent to customers informing them of their 30-day window to complete the migration of the specified servers. If you take no action, your server will be migrated for you.

To migrate your servers, during your migration window, simply perform a SOFT reboot, either from the Control Panel or by using the reboot API operation. The migration process will preserve all data and configuration settings.

During the 30-day migration window, performing a HARD reboot will reboot the server without triggering the migration.

You can see information about your migration window's open and close dates, use the Get Server Details operation, and look in the metadata section of the response for "FG2NG_self_migration_available_till" and "FG2NG_self_migration_available_from" key pairs. If your migration window has not been scheduled, you will not see these metadata keys.

For more information about the server migration see the Knowledge Center article: *First Generation to Next Generation cloud server migration FAQ*

This operation requires a request body.

This operation does not return a response body.

Example 3.21. Reboot Server: JSON Request

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

Example 3.22. Reboot Server: XML Request

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

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

3.3.2. Rebuild Server

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

Normal Response Code: 202

Error Response Codes: 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 3.23. Rebuild Server: JSON Request

```
{
    "rebuild" : {
        "imageId" : "115"
    }
}
```

Example 3.24. Rebuild Server: XML Request

```
<?xml version="1.0" encoding="UTF-8"?>
<rebuild xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    imageId="115"/>
```

This operation does not return a response body.

3.3.3. Resize Server

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

Normal Response Code: 202

Error Response Codes: 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 operation 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.

This operation does not return a response body.

Example 3.25. Resize Server: JSON Request

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

Example 3.26. Resize Server: XML Request

```
<?xml version="1.0" encoding="UTF-8"?>
<resize xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
 flavorId="3"/>
```

3.3.4. Confirm Resized Server

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

Normal Response Code: 204

Error Response Codes: 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.

This operation does not return a response body.

Example 3.27. Confirm Resize: JSON Request

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

Example 3.28. Confirm Resize: XML Request

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

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

3.3.5. Revert Resized Server

Verb	URI	Description
POST	/servers/*id*/action	Cancels and reverts a pending resize action.

Normal Response Code: 202

Error Response Codes: 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 3.29. Revert Resize: JSON Request

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

Example 3.30. Revert Resize: XML Request

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

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

This operation does not return a response body.

3.4. Flavors

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

3.4.1. List Flavors

Verb	URI	Description
GET	/flavors	Lists IDs and names for available flavors.
GET	/flavors/detail	Lists all details for available flavors.

Normal Response Codes: 200 and 203

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

This operation lists all available flavors with details.

This operation does not require a request body.

This operation returns a response body.

Example 3.31. List Flavors: JSON Response (detail)

```
{
    "flavors": [
        {
            "disk": 10,
            "id": 1,
            "name": "256 server",
            "ram": 256
        },
        {
            "disk": 20,
            "id": 2,
            "name": "512 server",
            "ram": 512
        },
        {
            "disk": 40,
            "id": 3,
            "name": "1GB server",
            "ram": 1024
        },
        {
            "disk": 80,
            "id": 4,
            "name": "2GB server",
            "ram": 2048
        },
        {
            "disk": 160,
            "id": 5,
            "name": "4GB server",
            "ram": 4096
        },
```

```
        {
            "disk": 320,
            "id": 6,
            "name": "8GB server",
            "ram": 8192
        },
        {
            "disk": 620,
            "id": 7,
            "name": "15.5GB server",
            "ram": 15872
        },
        {
            "disk": 1200,
            "id": 8,
            "name": "30GB server",
            "ram": 30720
        }
    ]
}
```

Example 3.32. List Flavors: XML Response (detail)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flavors xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
        <flavor disk="10" ram="256" name="256 server" id="1"/>
        <flavor disk="20" ram="512" name="512 server" id="2"/>
        <flavor disk="40" ram="1024" name="1GB server" id="3"/>
        <flavor disk="80" ram="2048" name="2GB server" id="4"/>
        <flavor disk="160" ram="4096" name="4GB server" id="5"/>
        <flavor disk="320" ram="8192" name="8GB server" id="6"/>
        <flavor disk="620" ram="15872" name="15.5GB server" id="7"/>
        <flavor disk="1200" ram="30720" name="30GB server" id="8"/>
</flavors>
```

3.4.2. Get Flavor Details

Verb	URI	Description
GET	/flavors/*id*	Lists details for the specified flavor.

Normal Response Codes: 200 and 203

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

This operation returns details for the specified flavor.

This operation does not require a request body.

Example 3.33. Get Flavor Details: JSON Response

```
{
    "flavor": {
        "disk": 20,
        "id": 2,
        "name": "512 server",
        "ram": 512
    }
}
```

Example 3.34. Get Flavor Details: XML Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<flavor xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    disk="20" ram="512" name="512 server" id="2"/>
```

3.5. 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.

3.5.1. List Images

Verb	URI	Description
GET	/images	Lists IDs and names for available images.
GET	/images/detail	Lists all details for available images.

Normal Response Codes: 200 and 203

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

This operation lists all images visible by the account.

The status for in-flight images is `SAVING`. The conditional progress element indicates the percentage completion and is from 0 to 100. 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 3.35. List Images: JSON Response (detail)


```
{
    "images": [
        {
            "id": 118,
            "name": "CentOS 6.0",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
            "id": 125,
            "name": "Ubuntu 12.04 LTS",
            "status": "ACTIVE",
            "updated": "2012-05-03T07:21:06-05:00"
        },
        {
            "id": 104,
            "name": "Debian 6 (Squeeze)",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
            "id": 107,
            "name": "FreeBSD 9.0",
            "status": "ACTIVE",
            "updated": "2012-04-24T10:48:08-05:00"
        },
        {
            "id": 24,
            "name": "Windows Server 2008 SP2 x64",
            "status": "ACTIVE",
```

```
            "updated": "2010-01-26T12:07:04-06:00"
        },
        {
            "id": 127,
            "name": "CentOS 6.3",
            "status": "ACTIVE",
            "updated": "2012-07-09T12:15:23-05:00"
        },
        {
            "id": 109,
            "name": "openSUSE 12",
            "status": "ACTIVE",
            "updated": "2011-11-03T06:28:56-05:00"
        },
        {
            "id": 85,
            "name": "Windows Server 2008 R2 x64",
            "status": "ACTIVE",
            "updated": "2010-01-26T12:07:17-06:00"
        },
        {
            "id": 110,
            "name": "Red Hat Enterprise Linux 5.5",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
            "id": 114,
            "name": "CentOS 5.6",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
            "id": 112,
            "name": "Ubuntu 10.04 LTS",
            "status": "ACTIVE",
            "updated": "2011-04-21T10:24:01-05:00"
        },
        {
            "id": 103,
            "name": "Debian 5 (Lenny)",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
            "id": 56,
            "name": "Windows Server 2008 SP2 x86 + SQL Server 2008 R2
Standard",
            "status": "ACTIVE",
            "updated": "2010-09-17T07:12:56-05:00"
        },
        {
            "id": 122,
            "name": "CentOS 6.2",
            "status": "ACTIVE",
            "updated": "2012-02-06T04:34:21-06:00"
        },
        {
            "id": 100,
            "name": "Arch 2012.08",
```

```
            "status": "ACTIVE",
            "updated": "2011-09-12T09:09:23-05:00"
        },
        {
            "id": 31,
            "name": "Windows Server 2008 SP2 x86",
            "status": "ACTIVE",
            "updated": "2010-01-26T12:07:44-06:00"
        },
        {
            "id": 91,
            "name": "Windows Server 2008 R2 x64 + SQL Server 2012 Standard",
            "status": "ACTIVE",
            "updated": "2012-04-24T16:44:01-05:00"
        },
        {
            "id": 111,
            "name": "Red Hat Enterprise Linux 6",
            "status": "ACTIVE",
            "updated": "2011-09-12T10:53:12-05:00"
        },
        {
            "id": 92,
            "name": "Windows Server 2008 R2 x64 + SQL Server 2012 Web",
            "status": "ACTIVE",
            "updated": "2012-04-24T16:44:01-05:00"
        },
        {
            "id": 57,
            "name": "Windows Server 2008 SP2 x64 + SQL Server 2008 R2
Standard",
            "status": "ACTIVE",
            "updated": "2010-09-17T07:16:25-05:00"
        },
        {
            "id": 120,
            "name": "Fedora 16",
            "status": "ACTIVE",
            "updated": "2012-01-03T04:39:05-06:00"
        },
        {
            "id": 86,
            "name": "Windows Server 2008 R2 x64 + SQL Server 2008 R2
Standard",
            "status": "ACTIVE",
            "updated": "2010-09-17T07:19:20-05:00"
        },
        {
            "id": 115,
            "name": "Ubuntu 11.04",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
            "id": 116,
            "name": "Fedora 15",
            "status": "ACTIVE",
            "updated": "2011-08-17T05:11:30-05:00"
        },
        {
```

```
            "id": 108,
            "name": "Gentoo 12.3",
            "status": "ACTIVE",
            "updated": "2011-11-01T08:32:30-05:00"
        },
        {
            "id": 126,
            "name": "Fedora 17",
            "status": "ACTIVE",
            "updated": "2012-05-29T17:11:45-05:00"
        },
        {
            "id": 121,
            "name": "CentOS 5.8",
            "status": "ACTIVE",
            "updated": "2012-05-04T10:51:28-05:00"
        },
        {
            "id": 89,
            "name": "Windows Server 2008 R2 x64 + SQL Server 2008 R2 Web",
            "status": "ACTIVE",
            "updated": "2011-10-04T08:39:34-05:00"
        },
        {
            "id": 119,
            "name": "Ubuntu 11.10",
            "status": "ACTIVE",
            "updated": "2011-11-03T08:55:15-05:00"
        }
    ]
}
```


Example 3.36. List Images: XML Response (detail)


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<images xmlns="http://docs.rackspacecloud.com/servers/api/v1.0">
 <image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
  name="CentOS 6.0" id="118"/>
 <image status="ACTIVE" updated="2012-05-03T07:21:06-05:00"
  name="Ubuntu 12.04 LTS" id="125"/>
 <image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
  name="Debian 6 (Squeeze)" id="104"/>
 <image status="ACTIVE" updated="2012-04-24T10:48:08-05:00"
  name="FreeBSD 9.0" id="107"/>
 <image status="ACTIVE" updated="2010-01-26T12:07:04-06:00"
  name="Windows Server 2008 SP2 x64" id="24"/>
 <image status="ACTIVE" updated="2012-07-09T12:15:23-05:00"
  name="CentOS 6.3" id="127"/>
 <image status="ACTIVE" updated="2011-11-03T06:28:56-05:00"
  name="openSUSE 12" id="109"/>
 <image status="ACTIVE" updated="2010-01-26T12:07:17-06:00"
  name="Windows Server 2008 R2 x64" id="85"/>
 <image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
  name="Red Hat Enterprise Linux 5.5" id="110"/>
 <image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
  name="CentOS 5.6" id="114"/>
 <image status="ACTIVE" updated="2011-04-21T10:24:01-05:00"
  name="Ubuntu 10.04 LTS" id="112"/>
 <image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
  name="Debian 5 (Lenny)" id="103"/>
```

```
<image status="ACTIVE" updated="2010-09-17T07:12:56-05:00"
 name="Windows Server 2008 SP2 x86 + SQL Server 2008 R2 Standard"
 id="56"/>
<image status="ACTIVE" updated="2012-02-06T04:34:21-06:00"
 name="CentOS 6.2" id="122"/>
<image status="ACTIVE" updated="2011-09-12T09:09:23-05:00"
 name="Arch 2012.08" id="100"/>
<image status="ACTIVE" updated="2010-01-26T12:07:44-06:00"
 name="Windows Server 2008 SP2 x86" id="31"/>
<image status="ACTIVE" updated="2012-04-24T16:44:01-05:00"
 name="Windows Server 2008 R2 x64 + SQL Server 2012 Standard"
 id="91"/>
<image status="ACTIVE" updated="2011-09-12T10:53:12-05:00"
 name="Red Hat Enterprise Linux 6" id="111"/>
<image status="ACTIVE" updated="2012-04-24T16:44:01-05:00"
 name="Windows Server 2008 R2 x64 + SQL Server 2012 Web"
 id="92"/>
<image status="ACTIVE" updated="2010-09-17T07:16:25-05:00"
 name="Windows Server 2008 SP2 x64 + SQL Server 2008 R2 Standard"
 id="57"/>
<image status="ACTIVE" updated="2012-01-03T04:39:05-06:00"
 name="Fedora 16" id="120"/>
<image status="ACTIVE" updated="2010-09-17T07:19:20-05:00"
 name="Windows Server 2008 R2 x64 + SQL Server 2008 R2 Standard"
 id="86"/>
<image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
 name="Ubuntu 11.04" id="115"/>
<image status="ACTIVE" updated="2011-08-17T05:11:30-05:00"
 name="Fedora 15" id="116"/>
<image status="ACTIVE" updated="2011-11-01T08:32:30-05:00"
 name="Gentoo 12.3" id="108"/>
<image status="ACTIVE" updated="2012-05-29T17:11:45-05:00"
 name="Fedora 17" id="126"/>
<image status="ACTIVE" updated="2012-05-04T10:51:28-05:00"
 name="CentOS 5.8" id="121"/>
<image status="ACTIVE" updated="2011-10-04T08:39:34-05:00"
 name="Windows Server 2008 R2 x64 + SQL Server 2008 R2 Web"
 id="89"/>
<image status="ACTIVE" updated="2011-11-03T08:55:15-05:00"
 name="Ubuntu 11.10" id="119"/>
</images>
```

3.5.2. Create Image

Verb	URI	Description
POST	/images	Creates a new image.

Normal Response Code: 202

Error Response Codes: 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 3.37. Create Image: JSON Request

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

Example 3.38. Create Image: JSON Response

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

Example 3.39. Create Image: XML Request

```
<?xml version="1.0" encoding="UTF-8"?>
<image
    xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
    name="Just in case"
    serverId="12"/>
```

Example 3.40. Create Image: XML Response

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

3.5.3. Get Image Details

Verb	URI	Description
GET	/images/*id*	Lists details for the specified image.

Normal Response Codes: 200 and 203

Error Response Codes: 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 3.5.1, “List Images” [47]

This operation does not require a request body.

Example 3.41. Get Image Details: JSON Response

```
{
    "image": {
        "created": "2011-11-03T08:55:15-05:00",
        "id": 119,
        "name": "Ubuntu 11.10",
        "status": "ACTIVE",
        "updated": "2011-11-03T08:55:15-05:00"
    }
}
```

Example 3.42. Get Image Details: XML Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<image xmlns="http://docs.rackspacecloud.com/servers/api/v1.0"
       status="ACTIVE" created="2011-11-03T08:55:15-05:00"
       updated="2011-11-03T08:55:15-05:00" name="Ubuntu 11.10"
       id="119"/>
```

3.5.4. Delete Image

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

Normal Response Code: 204

Error Response Codes: 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.

3.6. Backup Schedules

In addition to creating images on demand, you may also schedule periodic (daily and weekly) images through 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 3.3. Weekly Backup Schedule

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

Table 3.4. 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

3.6.1. List Backup Schedules

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

Normal Response Codes: 200 and 203

Error Response Codes: 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 3.43. List Backup Schedule: JSON Response

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

Example 3.44. List Backup Schedule: XML Response

```
<?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" />
```

3.6.2. Create / Update Backup Schedule

Verb	URI	Description
POST	/servers/*id*/backup_schedule	Enables a new backup schedule or updates an existing backup schedule for the specified server.

Normal Response Code: Accepted (202)

Error Response Codes: 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.

This operation does not return a response body.

Example 3.45. Update Backup Schedule: JSON Request

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

Example 3.46. Update Backup Schedule: XML Request

```
<?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" />
```

3.6.3. Disable Backup Schedule

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

Normal Response Code: 204

Error Response Codes: 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.

3.7. 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 can be a member of only one shared IP group.

3.7.1. List Shared IP Groups

Verb	URI	Description
GET	/shared_ip_groups	Lists IDs and names for shared IP groups.
GET	/shared_ip_groups/detail	Lists all details for shared IP groups.

Normal Response Codes: 200 and 203

Error Response Codes: 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 3.47. List Shared IP Groups: JSON Response (detail)

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

Example 3.48. List Shared IP Groups: XML Response (detail)

```
<?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>
```

3.7.2. Create Shared IP Group

Verb	URI	Description
POST	/shared_ip_groups	Creates a shared IP group.

Normal Response Code: 201

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

This operation creates a new shared IP group.

Note

When you create a shared IP group, you can add only one existing server to that group. That is your only chance to add an existing server. This is a technology limitation; IP group sharing depends on the servers being located on specific hypervisors and therefore, you cannot just assign random servers to the group. After you create a shared IP group, you add more servers to it when you create each server. However, you cannot add servers to the group retroactively.

All responses to requests for sharedIpGroup 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 generates a badRequest (400) fault.

This operation requires a request body.

This operation does not return a response body.

Example 3.49. Create Shared IP Group: JSON Request

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

Example 3.50. Create Shared IP Group: JSON Response

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

Example 3.51. Create Shared IP Group: XML Request

```
<?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 3.52. Create Shared IP Group Create: XML Response

```
<?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>
```

3.7.3. Get Shared IP Group Details

Verb	URI	Description
GET	/shared_ip_groups/*id*	Lists details for the specified shared IP group.

Normal Response Codes: 200 and 203

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

This operation returns details of the specified shared IP group.

Example 3.53. Get Shared IP Group Details: JSON Response

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

Example 3.54. Get Shared IP Group Details: XML Response

```
<?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>
```

This operation does not require a request body.

3.7.4. Delete Shared IP Group

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

Normal Response Code: 204

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

This operation deletes the specified shared IP group. This operation succeeds **only** if one of the following conditions is true:

- The group contains no active servers. For example, all servers were deleted.
- The group does not contain any servers that are actively sharing IPs.

This operation does not require a request body.

This operation does not contain a response body.