



Cloud Servers™ Developer Guide

API v0.9

This document is intended for software developers interested in developing applications using the Cloud Servers Application Programming Interface (API).

Table of Contents

Overview	1
Intended Audience	1
Concepts	2
Server	2
Flavor	2
Image	2
Snapshot	2
Backup Schedule	2
Reboot	3
Rebuild	3
Resize	3
Shared IP	3
Shared IP Group	3
General API Information	4
Authentication	4
Request/Response Content-Type	5
Content Compression	5
Persistent Connections	6
Paginated Collections	6
Caching	6
Efficient Polling with If-Modified-Since	7
System Thresholds	7
API Version	9
API Operations	12
Servers	12
List Servers	12
Create Server	14
Get Server Details	17
Update Server Name / Administrative Password	18

Delete Server.....	19
Server Actions	20
Reboot Server	20
Rebuild Server.....	21
Resize Server	21
Confirm Resized Server	22
Revert Resized Server	23
Initiate Snapshot.....	23
Share an IP Address	25
Unshare an IP Address	26
Flavors.....	26
List Flavors	27
Images.....	28
List Images	28
Backup Schedules.....	29
List Backup Schedules	31
Create / Update Backup Schedule	31
Delete Backup Schedule	32
Shared IP Groups.....	33
List Shared IP Groups	33
Create Shared IP Group.....	34
Get Shared IP Group Details.....	36
Delete Shared IP Group	37
Appendix.....	38
HTTP/1.1 Response Codes.....	38
Cloud Servers API Roadmap.....	39
Additional Resources.....	39

Overview

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

This document is a **PREVIEW**. It is intended to give customers and partners an opportunity to provide feedback into the design process for the Cloud Servers API. If you begin implementing against this early access specification, please recognize that it is still subject to change. Comments and feedback are welcomed at cloudserver_api@mosso.com.

Intended Audience

This guide is intended for software developers who want to develop applications using the Cloud Servers API. It assumes the reader has a general understanding of the Cloud Servers service and is familiar with:

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

Concepts

The Cloud Servers system has several key concepts that are important to understand:

Server

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

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.

Image

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

Snapshot

A snapshot is the action of creating an image from a running server. Snapshots are point in time file-level captures of servers and are useful for backup purposes or for creating custom images to launch new servers from.

Backup Schedule

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

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.

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.

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.

Shared IP

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.

Shared IP Group

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

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

Authentication

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

Request

The Rackspace Cloud Authentication Service is itself a RESTful web service. In order to authenticate, you must supply your username and API access key in the `X-Auth-User` and `X-Auth-Key` x-headers, respectively. Username is your common Rackspace Cloud username and your API access key can be obtained from the Rackspace Cloud Control Panel in the *Your Account | API Access* section.

Sample Request

```
GET /v1.1 HTTP/1.1
Host: auth.api.rackspacelcloud.com
X-Auth-User: jdoe
X-Auth-Key: a86850deb2742ec3cb41518e26aa2d89
```

Response

If authentication is successful, an HTTP status 204 No Content is returned with three cloud service headers, `X-Compute-Url`, `X-Storage-Url`, `X-CDN-Management-Url`, as well as `X-Auth-Token`. An HTTP status of 401 Unauthorized is returned if authentication fails. All operations against Cloud Servers should be performed against the URL specified in `X-Compute-Url` (which is dynamic and subject to change) and

must include the `X-Auth-Token` header as noted above. The URLs specified in `X-Storage-Url` and `X-CDN-Management-Url` are specific to the Cloud Files product and may be ignored for purposes of interacting with Cloud Servers.

Sample Response

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

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

Request/Response Content-Type

The Cloud Servers API supports both the JSON and XML data serialization formats. The format can be specified in requests using either the `Content-Type` header or via a URI query parameter. Responses will be serialized using the same scheme specified in the request. If no format is specified, JSON is the default. If conflicting formats are specified using both a `Content-Type` header and query parameter, the query parameter takes precedence.

Format	Content-Type Header	Query Parameter	Default
JSON	application/json	?format=json	Yes
XML	application/xml	?format=xml	No

Content Compression

Request and response body data may be encoded with gzip compression in order to

accelerate interactive performance of API calls and responses. This is controlled using the Accept-Encoding header on the request from the client and indicated by the Content-Encoding header in the server response. Unless the header is explicitly set, encoding defaults to disabled.

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

Persistent Connections

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

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

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

Paginated Collections

To reduce load on the service, some operations will return a limited number of items at a time. To navigate the collection, the parameters limit and offset can be set in the URI (e.g. ?limit=0&offset=0).

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. GETs 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 headers set:

Header	Description
Last Modified	Date and time when the entity was last updated in cache.
Expires	Date and time when the item expires from cache.

Efficient Polling with If-Modified-Since

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 If-Modified-Since header to check for changes since a previous request. The If-Modified-Since value may be taken from the HTTP/1.1 response header named “Last-Modified” from a previous GET request (this will be returned whether the previous response was cached or not). If nothing has changed since the If-Modified-Since value, a 304 Not Modified response will be returned. If data has changed, only the items changed since the If-Modified-Since value will be returned in the response.

Header Type	Name	Value	Example
HTTP/1.1 Request	If-Modified-Since	HTTP-date	06 Nov 1994 08:49:37 GMT
HTTP/1.1 Response	Last-Modified	HTTP-date	06 Nov 1994 08:49:37 GMT

System Thresholds

All accounts, by default, have a preconfigured set of thresholds to manage capacity and prevent abuse of the system. If the preconfigured limits are not suitable for your application, contact support@mosso.com to request an increase of the thresholds.

Default thresholds are as follows:

Verb	URI	Default Limit
POST	/*	10/min
PUT	/*	10/min

PUT	/servers	20/day
DELETE	/*	600/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. Depending on system conditions, the default rates are subject to change at any time. Applications should be designed to adjust to rate limit changes.

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

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

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 500

Request

This operation does not require a request body.

Sample XML Response

```
<limits>
  <limit>
    <verb>post</verb>
    <value>10</value>
    <metric>minute</metric>
    <uri>/*</uri>
  </limit>
  <limit>
    <verb>put</verb>
    <value>10</value>
    <metric>minute</metric>
    <uri>/*</uri>
  </limit>
  <limit>
    <verb>post</verb>
    <value>20</value>
    <metric>day</metric>
    <uri>/servers</uri>
```

```
</limit>
<limit>
  <verb>delete</verb>
  <value>600</value>
  <metric>minute</metric>
  <uri>/*</uri>
</limit>
</limits>
```

Sample JSON Response

```
{ "limits":
  { "limit":
    [ { "verb": "post",
        "value": "10",
        "metric": "minute",
        "uri": "/*" },
      { "verb": "put",
        "value": "10",
        "metric": "minute",
        "uri": "/*" },
      { "verb": "post",
        "value": "20",
        "metric": "day",
        "uri": "/servers" },
      { "verb": "delete",
        "value": "600",
        "metric": "minute",
        "uri": "/*" }
    ]
  }
}
```

API Version

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

When new API versions are released, Rackspace will work with developers and partners to ensure there is adequate time to migrate to the new version before older versions are deprecated.

Your application can programmatically determine available API versions by performing a GET on the root URL (i.e. with the version truncated) returned from the authentication system.

```
GET HTTP/1.1
Host: servers.api.rackspacecloud.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
```

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 500

Request

This operation does not require a request body

Sample XML Response

```
<versions>
  <version>
    <value>1.0</value>
    <status>obsolete</status>
    <docUrl>http://docs.rackspacecloud.com/cloudservers/cs-devguide-
v1.0.pdf<docUrl>
  </version>
  <version>
    <value>1.1</value>
    <status>current</status>
    <docUrl>http://docs.rackspacecloud.com/cloudservers/cs-devguide-
v1.1.pdf<docUrl>
  </version>
  <version>
    <value>1.2</value>
    <status>beta</status>
    <docUrl>http://docs.rackspacecloud.com/cloudservers/cs-devguide-
v1.2.pdf<docUrl>
  </version>
</versions>
```

Sample JSON Response

```
{ "versions":
  { "version":
    [ { "value": "1.0",
        "status": "obsolete",
        "docUrl": "http://docs.rackspacecloud.com/cloudservers/cs-devguide-1.0.pdf" },
      { "value": "1.1",
        "status": "current",
        "docUrl": "http://docs.rackspacecloud.com/cloudservers/cs-devguide-1.0.pdf" },
      { "value": "1.2",
        "status": "beta",
        "docUrl": "http://docs.rackspacecloud.com/cloudservers/cs-devguide-1.2.pdf" }
    ]
  }
}
```

API Operations

Servers

List Servers

Verb	URI	Description
GET	/servers	List servers

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 500

This operation provides a list of servers associated with your account. Servers that have been deleted are not included in this list. A maximum of 1,000 servers are returned at a time. To adjust the range of servers you are using, see the *Paginated Collections* section above. A list of server IDs is returned by default. To return a detailed list of servers, include the **?details** query parameter in the URI request (the sample responses below show a detailed response).

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

Note: *hostId* is unique PER ACCOUNT and is not globally unique.

Request

This operation does not require a request body.

Sample XML Response

```
<servers>
  <server>
    <serverId>1234</serverId>
    <serverName>sample-server</serverName>
    <imageId>1</imageId>
```

```
<flavorId>1</flavorId>
<hostId>9e107d9d372bb6826bd81d3542a419d6</hostId>
<ipAddrs>
  <publicInterface>67.23.10.132</publicInterface>
  <publicInterface>67.23.10.131</publicInterface>
  <privateInterface>10.176.42.16</privateInterface>
</ipAddrs>
<serverMetaData>
  <serverMetaDataKey>Server Label</serverMetaDataKey>
  <serverMetaDataValue>Web Head 1</serverMetaDataValue>
</serverMetaData>
<status>ACTIVE</status>
</server>
<server>
  <serverId>5678</serverId>
  <serverName>sample-server2</serverName>
  <imageId>1</imageId>
  <flavorId>1</flavorId>
  <hostId>e4d909c290d0fb1ca068ffaddf22cbd0</hostId>
  <ipAddrs>
    <publicInterface>67.23.10.133</publicInterface>
    <privateInterface>10.176.42.17</privateInterface>
  </ipAddrs>
  <serverMetaData>
    <serverMetaDataKey>Server Label</serverMetaDataKey>
    <serverMetaDataValue>DB 1</serverMetaDataValue>
  </serverMetaData>
  <serverMetaData>
    <serverMetaDataKey>My Image Version</serverMetaDataKey>
    <serverMetaDataValue>2.1</serverMetaDataValue>
  </serverMetaData>
  <status>BUILD</status>
  <progress>60</progress>
</server>
</servers>
```

Sample JSON Response

```
{"servers":
  {"server":
    [{"serverId": "1234",
      "serverName": "sample-server",
      "imageId": "1",
      "flavorId": "1",
      "hostId": "9e107d9d372bb6826bd81d3542a419d6",
      "ipAddrs":
        [{"publicInterface": "67.23.10.132"},
          {"publicInterface": "67.23.10.131"},
          {"privateInterface": "10.176.42.16"}]}
```

```

    ],
    "serverMetaData":
      {"serverMetaDataKey": "Server Label",
       "serverMetaDataValue": "Web Head 1"},
    "status": "ACTIVE"},
{"serverId": "5678",
 "serverName": "sample-server2",
 "imageId": "1",
 "flavorId": "1",
 "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
 "ipAddrs":
  [{"publicInterface": "67.23.10.133"},
   {"privateInterface": "10.176.42.17"}
 ],
 "serverMetaData":
  [{"serverMetaDataKey": "Server Label",
   "serverMetaDataValue": "DB 1"},
   {"serverMetaDataKey": "My Image Version",
    "serverMetaDataValue": "2.1"}
 ],
 "status": "BUILD",
 "progress": "60"}
]
}
}

```

Create Server

Verb	URI	Description
POST	/servers	Create a new server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 422, 500

This operation asynchronously provisions a new server. The progress of this operation depends on several factors including location of the requested image, network i/o, host load, and the selected flavor. The progress of the request can be checked by performing a GET on `/server/id`, which will return a progress element (0-100% completion). This element will only be returned when the *status* element is "BUILD".

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

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

Cloud servers can also be personalized by passing in custom data that is stored in the file system of the cloud server itself. You can personalize a server by specifying the *serverFilePath* and *serverFileContents* elements. This is useful, for example, for inserting ssh keys, setting configuration files, or storing data that you want to retrieve from within the instance itself. It is intended to provide a minimal amount of launch-time personalization. If significant customization is required, a custom image should be created via snapshot. The max size of the *serverFilePath* element is 255 bytes while the max size of *serverFileContents* is 10KB. The maximum number of file path/content pairs that can be supplied is 5. Any existing files that match the specified file will be overwritten. All files will have root and the root group as owner and group owner, respectively and will allow user and group read access only (-r--r-----).

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

If you intend to use a shared IP on the server being created and have no need for a separate public IP address, you may launch the server into a shared IP group and specify an IP address from that shared IP group to be used as its public IP. You can accomplish this by specifying the *ipAddr* element in your request. This is an optional element that is only valid if *sharedIpGroupId* is also supplied. If you need or want a separate public IP for this server, only supply *sharedGroupId* and once the server is started, you may then share an IP from the shared IP group using the `/server/id/actions/share_ip` operation.

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

Sample XML Request

```
<server>
  <serverName>new-server-test</serverName>
  <imageId>1</imageId>
  <flavorId>1</flavorId>
  <serverMetaData>
    <serverMetaDataKey>My Server Name</serverMetaDataKey>
    <serverMetaDataValue>Apache 1</serverMetaDataValue>
  </serverMetaData>
  <serverPersonality>
    <serverFilePath>/root/.ssh/authorized_keys</serverFilePath>
    <serverFileContents>ssh-dss AAAAB3NzaC1kc3MAAACBAJqkgr4 ...
FXpmTwyuqiHhyROsKaF3c3I4dfrOg== root@Test1</serverFileContents>
  </serverPersonality>
</server>
```

Sample JSON Request

```
{ "server":
  { "serverName": "new-server-test",
    "imageId": "1",
    "flavorId": "1",
    "serverMetaData":
      [ { "serverMetaDataKey": "My Server Name",
          "serverMetaDataValue": "Apache 1"
        }
      ],
    "serverPersonality":
      [ { "serverFilePath": "/root/.ssh/authorized_keys",
          "serverFileContents": "ssh-dss AAAAB3NzaC1kc3MAAACBAJqkgr4 ...
FXpmTwyuqiHhyROsKaF3c3I4dfrOg== root@Test1"
        }
      ]
  }
}
```

Sample XML Response

```
<server>
  <serverId>1235</serverId>
  <serverName>new-server-test</serverName>
  <imageId>1</imageId>
  <flavorId>1</flavorId>
  <hostId>e4d909c290d0fb1ca068ffaddf22cbd0</hostId>
  <ipAddrs>
    <publicInterface>67.23.10.138</publicInterface>
    <privateInterface>10.176.42.19</privateInterface>
  </ipAddrs>
  <progress>0</progress>
  <status>BUILD</status>
  <adminPass>GFflj9aP</adminPass>
  <serverMetaData>
```

```
<serverMetaDataKey>My Server Name</serverMetaDataKey>
<serverMetaDataValue>Apache 1</serverMetaDataValue>
</serverMetaData>
</server>
```

Sample JSON Response

```
{ "server":
  { "serverId": "1235",
    "serverName": "new-server-test",
    "imageId": "1",
    "flavorId": "1",
    "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
    "ipAddrs":
      [ { "publicInterface": "67.23.10.138"},
        { "privateInterface": "10.176.42.19"}
      ],
    "progress": "0",
    "status": "BUILD",
    "adminPass": "GFflj9aP",
    "serverMetaData":
      [ { "serverMetaDataKey": "My Server Name",
          "serverMetaDataValue": "Apache 1"
        }
      ]
  }
}
```

Get Server Details

Verb	URI	Description
GET	<i>/servers/id</i>	List details of the specified server

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 403, 404, 413, 500

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

Request

This operation does not require a request body.

Sample XML Response

```
<server>
  <serverId>1235</serverId>
  <serverName>new-server-test</serverName>
  <imageId>1</imageId>
  <flavorId>1</flavorId>
  <hostId>e4d909c290d0fb1ca068ffaddf22cbd0</hostId>
  <ipAddr>
    <publicInterface>67.23.10.138</publicInterface>
    <privateInterface>10.176.42.19</privateInterface>
  </ipAddr>
  <progress>20</progress>
  <status>BUILD</status>
  <serverMetaData>
    <serverMetaDataKey>My Server ID</serverMetaDataKey>
    <serverMetaDataValue>23</serverMetaDataValue>
  </serverMetaData>
</server>
```

Sample JSON Response

```
{ "server":
  { "serverId": "1235",
    "serverName": "new-server-test",
    "imageId": "1",
    "flavorId": "1",
    "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
    "ipAddr":
      [ { "publicInterface": "67.23.10.138" },
        { "privateInterface": "10.176.42.19" }
      ],
    "progress": "20",
    "status": "BUILD",
    "serverMetaData":
      [ { "serverMetaDataKey": "My Server ID",
          "serverMetaDataValue": "23" }
      ]
    }
}
```

Update Server Name / Administrative Password

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

		administrative password
--	--	-------------------------

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

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

Sample XML Request

```
<server>
  <serverName>updated-name</serverName>
  <adminPass>newpassword</adminPass>
</server>
```

Sample JSON Request

```
{"server":
  {"serverName": "updated-name",
   "adminPass": "newpassword"}
}
```

Response

This operation does not contain a response body.

Delete Server

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

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

This operation deletes a cloud server instance from the system. **When a server is deleted, all snapshots are also removed.**

Request

This operation does not require a request body.

Response

This operation does not contain a response body.

Server Actions

Reboot Server

Verb	URI	Description
POST	/servers/ <i>id</i> /actions/reboot	Reboot the specified server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

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.

Sample XML Request

```
<reboot>
  <type>soft</type>
</reboot>
```

Sample JSON Request

```
{"reboot":
  {"type": "soft"}
}
```

Response

This operation does not return a response body.

Rebuild Server

Verb	URI	Description
POST	/servers/ <i>id</i> /actions/rebuild	Rebuild the specified server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

The rebuild function removes all data on the server and replaces it with the specified image. *serverId* and IP addresses will remain the same. *imageId* is an optional argument. If it is not specified, the server is rebuilt with the original *imageId*.

Sample XML Request

```
<rebuild>
  <imageId>1</imageId>
</rebuild>
```

Sample JSON Request

```
{"rebuild":
  {"imageId": "1"}
}
```

Response

This operation does not return a response body.

Resize Server

Verb	URI	Description
POST	/servers/ <i>id</i> /actions/resize	Resize the specified server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

The resize function converts an existing server to a different flavor, in essence, scaling the server up or down. The original server is saved for a period of time to allow rollback if there is a problem. All resizes should be tested and explicitly confirmed, at which time the original server is removed. All resizes are automatically confirmed after 24 hours if they are not explicitly confirmed or reverted.

Sample XML Request

```
<resize>
  <flavorId>3</flavorId>
</resize>
```

Sample JSON Request

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

Response

This operation does not return a response body.

Confirm Resized Server

Verb	URI	Description
PUT	/servers/id/actions/resize	Confirm a pending resize action

Normal Response Code(s): 204

Error Response Code(s): 400, 401, 403, 404, 500

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.

Request

This operation does not require a request body.

Response

This operation does not return a response body.

Revert Resized Server

Verb	URI	Description
DELETE	/servers/id/actions/resize	Cancel and revert a pending resize action

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

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.

Request

This operation does not require a request body.

Response

This operation does not return a response body.

Initiate Snapshot

Verb	URI	Description
POST	/servers/id/actions/snapshot	Create a snapshot of the specified server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 500

This operation creates a new snapshot for the given server ID. Once complete, a new image will be available that can be used to rebuild or create servers. Specifying the same image name as an existing snapshot/image replaces the image. The image creation status can be queried by performing a GET on /images and examining the *status* and *progress* elements.

Note: *At present, snapshots are an asynchronous operation, so coordinating the snap with data quiescence, etc. is currently not possible.*

Sample XML Request

```
<snapshot>
  <imageName>Just in case</imageName>
</snapshot>
```

Sample JSON Request

```
{"snapshot":
  {"imageName": "Just in case"}
}
```

Sample XML Response

```
<image>
  <imageId>377</imageId>
  <imageName>Just in case</imageName>
  <timeStamp>2009-10-10T12:00:00Z</timeStamp>
  <status>SAVING</status>
  <progress>0</progress>
</image>
```

Sample JSON Response

```
{"image":
  {"imageId": "377",
    "imageName": "Just in case",
    "timeStamp": "2009-10-10T12:00:00Z",
    "status": "SAVING",
    "progress": "0"}
}
```

```
}

```

Share an IP Address

Verb	URI	Description
POST	/servers/id/actions/share_ip	Share an IP address to the specified server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 422, 500

This operation shares an IP from an existing server in the specified shared IP group to another specified server in the same group. Specifically, this operation modifies cloud network restrictions to allow IP traffic for the given IP to/from the server specified, but does not bind the IP to the server itself. A heartbeat facility (e.g. keepalived) should then be used within the servers to perform health checks and manage IP failover.

Sample XML Request

```
<shareIp>
  <sharedIpGroupId>1234</sharedIpGroupId>
  <ipAddr>67.23.10.138</ipAddr>
</shareIp>
```

Sample JSON Request

```
{"shareIp":
  {"sharedIpGroupId": "1234",
   "ipAddr": "67.23.10.138"}
}
```

Response

This operation does not return a response body.

Unshare an IP Address

Verb	URI	Description
POST	/servers/ <i>id</i> /actions/unshare_ip	Remove a shared IP address from the specified server

Normal Response Code(s): 202

Error Response Code(s): 400, 401, 403, 404, 422, 500

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

Sample XML Request

```
<unshareIp>
  <ipAddr>67.23.10.138</ipAddr>
</unshareIp>
```

Sample JSON Request

```
{ "unshareIp":
  { "ipAddr": "67.23.10.138" }
}
```

Response

This operation does not return a response body.

Flavors

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.

List Flavors

Verb	URI	Description
GET	/flavors	List the available flavors, with details of each

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 500

This operation will list all available flavors with details.

Request

This operation does not require a request body.

Sample XML Response

```
<flavors>
  <flavor>
    <flavorId>1</flavorId>
    <flavorName>256 MB Server</flavorName>
    <ram>256</ram>
    <disk>10</disk>
  </flavor>
  <flavor>
    <flavorId>2</flavorId>
    <flavorName>512 MB Server</flavorName>
    <ram>512</ram>
    <disk>20</disk>
  </flavor>
</flavors>
```

Sample JSON Response

```
{"flavors":
  {"flavor":
    [{"flavorId": "1",
      "flavorName": "256 MB Server",
      "ram": "256",
      "disk": "10"}],
```

```

    {"flavorId": "2",
      "flavorName": "512 MB Server",
      "ram": "512",
      "disk": "20"}
  ]
}

```

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 via server snapshots.

List Images

Verb	URI	Description
GET	/images	List the available images, with details of each

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 500

This operation will list all images visible by the account. In-flight snapshots will have the *status* element set to *SAVING* and the conditional *progress* element (0-100% completion) will also be returned. Images with an *ACTIVE* status are available for install.

Request

This operation does not require a request body.

Sample XML Response

```

<images>
  <image>

```

```
<imageId>1</imageId>
<imageName>CentOS 5.2</imageName>
<timeStamp>2010-10-10T12:00:00Z</timeStamp>
<status>ACTIVE</status>
</image>
<image>
  <imageId>743</imageId>
  <imageName>My Server Backup</imageName>
  <timeStamp>2010-10-10T12:00:00Z</timeStamp>
  <status>SAVING</status>
  <progress>80</progress>
</image>
</images>
```

Sample JSON Response

```
{ "images":
  { "image":
    [ { "imageId": "1",
        "imageName": "CentOS 5.2",
        "timeStamp": "2010-10-10T12:00:00Z",
        "status": "ACTIVE" },
      { "imageId": "743",
        "imageName": "My Server Backup",
        "timeStamp": "2010-10-10T12:00:00Z",
        "status": "SAVING",
        "progress": "80" }
    ]
  }
}
```

Backup Schedules

In addition to on-demand snapshots, you may also schedule periodic (daily and weekly) snapshots via a backup schedule. The daily and weekly snapshots 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.

Weekly Backup Schedule

Value	Day
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Daily Backup Schedule

Value	Hour Range
1	0000-0200
2	0200-0400
3	0400-0600
4	0600-0800
5	0800-1000
6	1000-1200
7	1200-1400
8	1400-1600
9	1600-1800
10	1800-2000
11	2000-2200
12	2200-0000

List Backup Schedules

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

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 403, 404, 500

This operation lists the backup schedule for the specified server. If a server does not have a backup schedule enabled, the *status* element will be “disabled” and weekly/daily elements will not be provided in the response.

Request

This operation does not require a request body.

Sample XML Response

```
<backupSchedule>
  <status>enabled</status>
  <weekly>3</weekly>
  <daily>5</daily>
</backupSchedule>
```

Sample JSON Response

```
{"backupSchedule":
  {"status": "enabled", "weekly": "3", "daily": "5"}
}
```

Create / Update Backup Schedule

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

Normal Response Code(s): 204

Error Response Code(s): 400, 401, 403, 404, 500

This operation creates a new backup schedule or updates an existing backup schedule for the specified server.

Sample XML Request

```
<backupSchedule>
  <status>enabled</status>
  <weekly>3</weekly>
  <daily>5</daily>
</backupSchedule>
```

Sample JSON Request

```
{"backupSchedule":
  {"status": "enabled", "weekly": "3", "daily": "5"}
}
```

Response

This operation does not return a response body.

Delete Backup Schedule

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

Normal Response Code(s): 204

Error Response Code(s): 400, 401, 403, 404, 500

This operation deletes the backup schedule for the specified server.

Request

This operation does not require a request body.

Response

This operation does not return a response body.

Shared IP Groups

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

List Shared IP Groups

Verb	URI	Description
GET	/shared_ip_groups	List shared ip groups

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 500

This operation provides a list of shared IP groups associated with your account. Shared IP groups that have been deleted are not included in this list. A maximum of 1,000 shared IP groups are returned at a time. To adjust the range of shared IP groups you are using, see the *paginated collections* section above. A list of shared IP group IDs is returned by default. To return a detailed list of shared IP groups, include the **?details** query parameter in the URI request (the sample responses below show a detailed response).

Request

This operation does not require a request body.

Sample XML Response

```
<sharedIpGroups>
  <sharedIpGroup>
```

```
<sharedIpGroupId>1234</sharedIpGroupId>
<sharedIpGroupName>Shared IP Group 1</sharedIpGroupName>
<servers>
  <serverId>422</serverId>
  <serverId>3445</serverId>
</servers>
</sharedIpGroup>
<sharedIpGroup>
  <sharedIpGroupId>5678</sharedIpGroupId>
  <sharedIpGroupName>Shared IP Group 2</sharedIpGroupName>
  <servers>
    <serverId>23203</serverId>
    <serverId>2456</serverId>
    <serverId>9891</serverId>
  </servers>
</sharedIpGroup>
</sharedIpGroups>
```

Sample JSON Request

```
{ "sharedIpGroups":
  { "sharedIpGroup":
    [ { "sharedIpGroupId": "1234",
      "sharedIpGroupName": "Shared IP Group 1",
      "servers":
        [ { "serverId": "422"},
          { "serverId": "3445"}
        ]
      },
    { "sharedIpGroupId": "5678",
      "sharedIpGroupName": "Shared IP Group 2",
      "servers":
        [ { "serverId": "23203"},
          { "serverId": "2456"},
          { "serverId": "9891"}
        ]
      }
    ]
  }
}
```

Create Shared IP Group

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

Normal Response Code(s): 201

Error Response Code(s): 400, 401, 403, 404, 422, 500

This operation creates a new shared IP group. The group can be created empty or can be initially populated with a single server by specifying the optional *serverId* element.

Sample XML Request

```
<sharedIpGroup>
  <sharedIpGroupName>Web Cluster 1</sharedIpGroupName>
  <serverId>8361</serverId>
</sharedIpGroup>
```

Sample JSON Request

```
{"sharedIpGroup":
  {"sharedIpGroupName": "Web Cluster 1",
   "serverId": "8361"}
}
```

Sample XML Response

```
<sharedIpGroup>
  <sharedIpGroupId>235</sharedIpGroupId>
  <sharedIpGroupName>Web Cluster 1</sharedIpGroupName>
  <servers>
    <serverId>8361</serverId>
  </servers>
</sharedIpGroup>
```

Sample JSON Response

```
{"sharedIpGroup":
  {"sharedIpGroupId": "235",
   "sharedIpGroupName": "Web Cluster 1",
   "servers":
    [{"serverId": "8361"}]
  }
}
```

Get Shared IP Group Details

Verb	URI	Description
GET	/shared_ip_groups/ <i>id</i>	List details of the specified shared IP group

Normal Response Code(s): 200, 203

Error Response Code(s): 400, 401, 403, 404, 500

This operation returns details of the specified shared IP group.

Request

This operation does not require a request body.

Sample XML Response

```
<sharedIpGroup>
  <sharedIpGroupId>1234</sharedIpGroupId>
  <sharedIpGroupName>Shared IP Group 1</sharedIpGroupName>
  <servers>
    <serverId>422</serverId>
    <serverId>3445</serverId>
  </servers>
</sharedIpGroup>
```

Sample JSON Response

```
{ "sharedIpGroup":
  { "sharedIpGroupId": "1234",
    "sharedIpGroupName": "Shared IP Group 1",
    "servers":
      [ { "serverId": "422"},
        { "serverId": "3445"}
      ]
  }
}
```

Delete Shared IP Group

Verb	URI	Description
DELETE	/shared_ip_groups/ <i>id</i>	Delete the specified shared IP group

Normal Response Code(s): 204

Error Response Code(s): 400, 401, 403, 404, 500

This operation deletes the specified shared IP group. This operation will ONLY succeed if 1) there are no active servers in the group (i.e. they have all been terminated) or 2) no servers in the group are actively sharing IPs.

Request

This operation does not require a request body.

Response

This operation does not contain a response body.

Appendix

HTTP/1.1 Response Codes

The Cloud Servers API may return any of the HTTP/1.1 response codes defined by [RFC-2616 Section 10](#). The API uses the following return codes as described:

Code	Description
200 OK	Fresh authoritative content returned.
201 Created	A POST was accepted, and completed synchronously. The body of the response will contain the id of the created entity.
202 Accepted	An asynchronous operation was started. Where possible, the body of the response will contain an id that can be used to poll the status of the request for completion.
203 Cached	Cached content was returned. If possible, the server will return a Retry-After header indicating when fresh content will be available.
204 No Content	The request was accepted and processed, but did not need to return a response body.
304 Not Modified	Indicates that the requested resource's status has not changed since the specified If-Modified-Since date.
400 Bad Request	There was a syntax error in the request. Do not repeat the request without adjusting the input.
401 Unauthorized	The X-Auth-Token is not valid or has expired. Re-authenticate to obtain a fresh token.
403 Forbidden	Access is denied for the given request. Check your X-Auth-Token header. The token may have expired.
404 Not Found	The server has not found anything matching the Request URI.
413 Request Entity Too Large	The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request. If the condition is temporary, the server will include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.
413 Rate Control	The server is refusing to process the request because the client request rate exceeds a configured limit for the given account and action type. The server will include a Retry-After header field to indicate that it is temporary and after what time the client MAY try

	again.
422 Unprocessable Entity	The server understands the content type of the request entity and the syntax of the request entity is correct, but was unable to process the contained instructions. For example, this error condition may occur if an XML request body contains well-formed, but semantically erroneous, XML instructions.
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.

Cloud Servers API Roadmap

The following capabilities are planned for the subsequent Cloud Servers API release:

- **Backup Integration with Cloud Files** - Backups are stored in your Cloud Files account. You make take unlimited on-demand snapshots and pricing is completely utility based - you pay only for the space you consume on Cloud Files. Backups can be downloaded and are also decoupled from servers so they persist after a server is terminated.
- **Custom Image Upload** - Currently, you can customize an image by starting with a Rackspace provided default, installing applications, making desired configuration changes, and taking a snapshot. In this release, we enable you to upload your own custom images to Cloud Files and register them with Cloud Servers for deployment.
- **Image Sharing** - You can mark images as public or take advantage of image access groups to delegate image access to a specific group of Cloud Servers users.

Additional Resources

The official support channels (phone, chat, email, forums, and knowledge base articles) for Cloud Servers are available at the Rackspace Cloud website:

<http://www.mosso.com>.

Interested users can also follow updates/announcements via twitter at <http://www.twitter.com/cloudservers>