ADMIN OPERATIONS

An admin API request will be done on a URI that starts with the configurable 'admin' resource entry point. Authorization for the admin API duplicates the S3 authorization mechanism. Some operations require that the user holds special administrative capabilities. The response entity type (XML or JSON) may be specified as the 'format' option in the request and defaults to JSON if not specified.

GET USAGE

Request bandwidth usage information.

Note: this feature is disabled by default, can be enabled by setting `rgw enable usage log = true` in the appropriate section of ceph.conf. For changes in ceph.conf to take effect, radosgw process restart is needed.

caps:	usage=read

SYNTAX

```
GET /{admin}/usage?format=json HTTP/1.1
Host: {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user for which the information is requested. If not specified will apply to all users.
Type:	String
Example:	`foo_user`
Required:	No

`start`

Description:	Date and (optional) time that specifies the start time of the requested data.
Type:	String
Example:	`2012-09-25 16:00:00`
Required:	No

`end`

Description:	Date and (optional) time that specifies the end time of the requested data (non-inclusive).
Type:	String
Example:	`2012-09-25 16:00:00`
Required:	No

`show-entries`

Description:	Specifies whether data entries should be returned.
Type:	Boolean
Example:	True [True]
Required:	No

`show-summary`

Description:	Specifies whether data summary should be returned.
Type:	Boolean
Example:	True [True]
Required:	No

RESPONSE ENTITIES

If successful, the response contains the requested information.

`usage`

Description: A container for the usage information.
Type: Container

`entries`

Description: A container for the usage entries information.
Type: Container

`user`

Description: A container for the user data information.
Type: Container

`owner`

Description: The name of the user that owns the buckets.
Type: String

`bucket`

Description: The bucket name.
Type: String

`time`

Description: Time lower bound for which data is being specified (rounded to the beginning of the first relevant hour).
Type: String

`epoch`

Description: The time specified in seconds since 1/1/1970.
Type: String

`categories`

Description: A container for stats categories.
Type: Container

`entry`

Description: A container for stats entry.
Type: Container

`category`

Description: Name of request category for which the stats are provided.
Type: String

`bytes_sent`

Description: Number of bytes sent by the RADOS Gateway.
Type: Integer

`bytes_received`

Description: Number of bytes received by the RADOS Gateway.
Type: Integer

`ops`

Description:	Number of operations.
Type:	Integer

`successful_ops`

Description:	Number of successful operations.
Type:	Integer

`summary`

Description:	A container for stats summary.
Type:	Container

`total`

Description:	A container for stats summary aggregated total.
Type:	Container

SPECIAL ERROR RESPONSES

TBD.

TRIM USAGE

Remove usage information. With no dates specified, removes all usage information.

Note: this feature is disabled by default, can be enabled by setting `rgw enable usage log = true` in the appropriate section of ceph.conf. For changes in ceph.conf to take effect, radosgw process restart is needed.

caps: usage=write

SYNTAX

```
DELETE /{admin}/usage?format=json HTTP/1.1
Host: {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user for which the information is requested. If not specified will apply to all users.
Type:	String
Example:	`foo_user`
Required:	No

`start`

Description:	Date and (optional) time that specifies the start time of the requested data.
Type:	String
Example:	`2012-09-25 16:00:00`
Required:	No

`end`

Description:	Date and (optional) time that specifies the end time of the requested data (none inclusive).
Type:	String
Example:	`2012-09-25 16:00:00`
Required:	No

`remove-all`

Description:	Required when uid is not specified, in order to acknowledge multi user data removal.
Type:	Boolean
Example:	True [False]
Required:	No

SPECIAL ERROR RESPONSES

TBD.

GET USER INFO

Get user information.

caps: users=read

SYNTAX

```
GET /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user for which the information is requested.
Type:	String
Example:	`foo_user`
Required:	Yes

RESPONSE ENTITIES

If successful, the response contains the user information.

`user`

Description:	A container for the user data information.
Type:	Container

`user_id`

Description:	The user id.
Type:	String
Parent:	`user`

`display_name`

Description:	Display name for the user.
Type:	String
Parent:	`user`

`suspended`

Description:	True if the user is suspended.
Type:	Boolean
Parent:	`user`

`max_buckets`

Description:	The maximum number of buckets to be owned by the user.
Type:	Integer

Parent:	`user`

`subusers`

Description:	Subusers associated with this user account.
Type:	Container
Parent:	`user`

`keys`

Description:	S3 keys associated with this user account.
Type:	Container
Parent:	`user`

`swift_keys`

Description:	Swift keys associated with this user account.
Type:	Container
Parent:	`user`

`caps`

Description:	User capabilities.
Type:	Container
Parent:	`user`

SPECIAL ERROR RESPONSES

None.

CREATE USER

Create a new user. By default, a S3 key pair will be created automatically and returned in the response. If only one of `access-key` or `secret-key` is provided, the omitted key will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If `access-key` is specified and refers to an existing key owned by the user then it will be modified.

New in version Luminous.

A `tenant` may either be specified as a part of uid or as an additional request param.

caps:	users=write

SYNTAX

```
PUT /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID to be created.
Type:	String
Example:	`foo_user`
Required:	Yes

A tenant name may also specified as a part of `uid`, by following the syntax `tenant$user`, refer to Multitenancy for more details.

`display-name`

Description:	The display name of the user to be created.
Type:	String
Example:	`foo user`
Required:	Yes

`email`

Description:	The email address associated with the user.
Type:	String
Example:	`foo@bar.com`
Required:	No

`key-type`

Description:	Key type to be generated, options are: swift, s3 (default).
Type:	String
Example:	`s3` [`s3`]
Required:	No

`access-key`

Description:	Specify access key.
Type:	String
Example:	`ABCD0EF12GHIJ2K34LMN`
Required:	No

`secret-key`

Description:	Specify secret key.
Type:	String
Example:	`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`
Required:	No

`user-caps`

Description:	User capabilities.
Type:	String
Example:	`usage=read, write; users=read`
Required:	No

`generate-key`

Description:	Generate a new key pair and add to the existing keyring.
Type:	Boolean
Example:	True [True]
Required:	No

`max-buckets`

Description:	Specify the maximum number of buckets the user can own.
Type:	Integer
Example:	500 [1000]
Required:	No

`suspended`

Description:	Specify whether the user should be suspended.
Type:	Boolean
Example:	False [False]
Required:	No

New in version Jewel.

`tenant`

Description:	the Tenant under which a user is a part of.
Type:	string
Example:	tenant1
Required:	No

RESPONSE ENTITIES

If successful, the response contains the user information.

`user`

Description:	A container for the user data information.
Type:	Container

`tenant`

Description:	The tenant which user is a part of.
Type:	String
Parent:	`user`

`user_id`

Description:	The user id.
Type:	String
Parent:	`user`

`display_name`

Description:	Display name for the user.
Type:	String
Parent:	`user`

`suspended`

Description:	True if the user is suspended.
Type:	Boolean
Parent:	`user`

`max_buckets`

Description:	The maximum number of buckets to be owned by the user.
Type:	Integer
Parent:	`user`

`subusers`

Description:	Subusers associated with this user account.
Type:	Container
Parent:	`user`

`keys`

Description:	S3 keys associated with this user account.
Type:	Container
Parent:	`user`

`swift_keys`

Description:	Swift keys associated with this user account.
Type:	Container
Parent:	`user`

`caps`

Description:	User capabilities.
Type:	Container
Parent:	`user`

SPECIAL ERROR RESPONSES

`UserExists`

Description:	Attempt to create existing user.
Code:	409 Conflict

`InvalidAccessKey`

Description:	Invalid access key specified.
Code:	400 Bad Request

`InvalidKeyType`

Description:	Invalid key type specified.
Code:	400 Bad Request

`InvalidSecretKey`

Description:	Invalid secret key specified.
Code:	400 Bad Request

`InvalidKeyType`

Description:	Invalid key type specified.
Code:	400 Bad Request

`KeyExists`

Description:	Provided access key exists and belongs to another user.
Code:	409 Conflict

`EmailExists`

Description:	Provided email address exists.
Code:	409 Conflict

`InvalidCapability`

Description:	Attempt to grant invalid admin capability.
Code:	400 Bad Request

MODIFY USER

Modify a user.

caps: users=write

SYNTAX

```
POST /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID to be modified.
Type:	String
Example:	`foo_user`
Required:	Yes

`display-name`

Description:	The display name of the user to be modified.
Type:	String
Example:	`foo user`
Required:	No

`email`

Description:	The email address to be associated with the user.
Type:	String
Example:	`foo@bar.com`
Required:	No

`generate-key`

Description:	Generate a new key pair and add to the existing keyring.
Type:	Boolean
Example:	True [False]
Required:	No

`access-key`

Description:	Specify access key.
Type:	String
Example:	`ABCD0EF12GHIJ2K34LMN`
Required:	No

`secret-key`

Description:	Specify secret key.
Type:	String
Example:	`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`
Required:	No

`key-type`

Description:	Key type to be generated, options are: swift, s3 (default).
Type:	String
Example:	`s3`
Required:	No

`user-caps`

Description:	User capabilities.
Type:	String
Example:	`usage=read, write; users=read`
Required:	No

`max-buckets`

Description:	Specify the maximum number of buckets the user can own.
Type:	Integer
Example:	500 [1000]
Required:	No

`suspended`

Description:	Specify whether the user should be suspended.

Type:	Boolean
Example:	False [False]
Required:	No

RESPONSE ENTITIES

If successful, the response contains the user information.

`user`

Description:	A container for the user data information.
Type:	Container

`user_id`

Description:	The user id.
Type:	String
Parent:	`user`

`display_name`

Description:	Display name for the user.
Type:	String
Parent:	`user`

`suspended`

Description:	True if the user is suspended.
Type:	Boolean
Parent:	`user`

`max_buckets`

Description:	The maximum number of buckets to be owned by the user.
Type:	Integer
Parent:	`user`

`subusers`

Description:	Subusers associated with this user account.
Type:	Container
Parent:	`user`

`keys`

Description:	S3 keys associated with this user account.
Type:	Container
Parent:	`user`

`swift_keys`

Description:	Swift keys associated with this user account.
Type:	Container
Parent:	`user`

`caps`

Description:	User capabilities.
Type:	Container
Parent:	`user`

SPECIAL ERROR RESPONSES

`InvalidAccessKey`

Description:	Invalid access key specified.
Code:	400 Bad Request

`InvalidKeyType`

Description:	Invalid key type specified.
Code:	400 Bad Request

`InvalidSecretKey`

Description:	Invalid secret key specified.
Code:	400 Bad Request

`KeyExists`

Description:	Provided access key exists and belongs to another user.
Code:	409 Conflict

`EmailExists`

Description:	Provided email address exists.
Code:	409 Conflict

`InvalidCapability`

Description:	Attempt to grant invalid admin capability.
Code:	400 Bad Request

REMOVE USER

Remove an existing user.

caps: users=write

SYNTAX

```
DELETE /{admin}/user?format=json HTTP/1.1
Host: {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID to be removed.
Type:	String
Example:	`foo_user`
Required:	Yes.

`purge-data`

Description:	When specified the buckets and objects belonging to the user will also be removed.
Type:	Boolean
Example:	True
Required:	No

RESPONSE ENTITIES

None

SPECIAL ERROR RESPONSES

None.

CREATE SUBUSER

Create a new subuser (primarily useful for clients using the Swift API). Note that in general for a subuser to be useful, it must be granted permissions by specifying `access`. As with user creation if `subuser` is specified without `secret`, then a secret key will be automatically generated.

caps: users=write

SYNTAX

```
PUT /{admin}/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID under which a subuser is to be created.
Type:	String
Example:	`foo_user`
Required:	Yes

`subuser`

Description:	Specify the subuser ID to be created.
Type:	String
Example:	`sub_foo`
Required:	Yes

`secret-key`

Description:	Specify secret key.
Type:	String
Example:	`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`
Required:	No

`key-type`

Description:	Key type to be generated, options are: swift (default), s3.
Type:	String
Example:	`swift` [`swift`]
Required:	No

`access`

Description:	Set access permissions for sub-user, should be one of `read, write, readwrite, full`.
Type:	String
Example:	`read`
Required:	No

`generate-secret`

Description:	Generate the secret key.
Type:	Boolean
Example:	True [False]

Required:	No

RESPONSE ENTITIES

If successful, the response contains the subuser information.

`subusers`

Description:	Subusers associated with the user account.
Type:	Container

`id`

Description:	Subuser id.
Type:	String
Parent:	`subusers`

`permissions`

Description:	Subuser access to user account.
Type:	String
Parent:	`subusers`

SPECIAL ERROR RESPONSES

`SubuserExists`

Description:	Specified subuser exists.
Code:	409 Conflict

`InvalidKeyType`

Description:	Invalid key type specified.
Code:	400 Bad Request

`InvalidSecretKey`

Description:	Invalid secret key specified.
Code:	400 Bad Request

`InvalidAccess`

Description:	Invalid subuser access specified.
Code:	400 Bad Request

MODIFY SUBUSER

Modify an existing subuser

caps:	users=write

SYNTAX

```
POST /{admin}/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID under which the subuser is to be modified.
Type:	String
Example:	`foo_user`
Required:	Yes

`subuser`

Description:	The subuser ID to be modified.
Type:	String
Example:	`sub_foo`
Required:	Yes

`generate-secret`

Description:	Generate a new secret key for the subuser, replacing the existing key.
Type:	Boolean
Example:	True [False]
Required:	No

`secret`

Description:	Specify secret key.
Type:	String
Example:	`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`
Required:	No

`key-type`

Description:	Key type to be generated, options are: swift (default), s3 .
Type:	String
Example:	`swift` [`swift`]
Required:	No

`access`

Description:	Set access permissions for sub-user, should be one of `read, write, readwrite, full`.
Type:	String
Example:	`read`
Required:	No

RESPONSE ENTITIES

If successful, the response contains the subuser information.

`subusers`

Description:	Subusers associated with the user account.
Type:	Container

`id`

Description:	Subuser id.
Type:	String
Parent:	`subusers`

`permissions`

Description:	Subuser access to user account.
Type:	String
Parent:	`subusers`

SPECIAL ERROR RESPONSES

`InvalidKeyType`

Description:	Invalid key type specified.
Code:	400 Bad Request

`InvalidSecretKey`

Description:	Invalid secret key specified.
Code:	400 Bad Request

`InvalidAccess`

Description:	Invalid subuser access specified.
Code:	400 Bad Request

REMOVE SUBUSER

Remove an existing subuser

caps: users=write

SYNTAX

```
DELETE /{admin}/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID under which the subuser is to be removed.
Type:	String
Example:	`foo_user`
Required:	Yes

`subuser`

Description:	The subuser ID to be removed.
Type:	String
Example:	`sub_foo`
Required:	Yes

`purge-keys`

Description:	Remove keys belonging to the subuser.
Type:	Boolean
Example:	True [True]
Required:	No

RESPONSE ENTITIES

None.

SPECIAL ERROR RESPONSES

None.

CREATE KEY

Create a new key. If a `subuser` is specified then by default created keys will be swift type. If only one of `access-key` or `secret-key` is provided the committed key will be automatically generated, that is if only `secret-key` is specified then `access-key` will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If `access-key` is specified and refers to an existing key owned by the user then it will be modified. The response is a container listing all keys of the same type as the key created. Note that when creating a swift key, specifying the option `access-key` will have no effect. Additionally, only one swift key may be held by each user or subuser.

caps:	users=write

SYNTAX

```
PUT /{admin}/user?key&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID to receive the new key.
Type:	String
Example:	`foo_user`
Required:	Yes

`subuser`

Description:	The subuser ID to receive the new key.
Type:	String
Example:	`sub_foo`
Required:	No

`key-type`

Description:	Key type to be generated, options are: swift, s3 (default).
Type:	String
Example:	s3 [s3]
Required:	No

`access-key`

Description:	Specify the access key.
Type:	String
Example:	`AB01C2D3EF45G6H7IJ8K`
Required:	No

`secret-key`

Description:	Specify the secret key.
Type:	String
Example:	`0ab/CdeFGhij1klmnopqRSTUv1WxyZabcDEFgHij`
Required:	No

`generate-key`

Description:	Generate a new key pair and add to the existing keyring.
Type:	Boolean
Example:	True [True]
Required:	No

RESPONSE ENTITIES

`keys`

Description: Keys of type created associated with this user account.
Type: Container

`user`

Description: The user account associated with the key.
Type: String
Parent: keys

`access-key`

Description: The access key.
Type: String
Parent: keys

`secret-key`

Description: The secret key
Type: String
Parent: keys

SPECIAL ERROR RESPONSES

`InvalidAccessKey`

Description: Invalid access key specified.
Code: 400 Bad Request

`InvalidKeyType`

Description: Invalid key type specified.
Code: 400 Bad Request

`InvalidSecretKey`

Description: Invalid secret key specified.
Code: 400 Bad Request

`InvalidKeyType`

Description: Invalid key type specified.
Code: 400 Bad Request

`KeyExists`

Description: Provided access key exists and belongs to another user.
Code: 409 Conflict

REMOVE KEY

Remove an existing key.

caps: users=write

SYNTAX

```
DELETE /{admin}/user?key&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`access-key`

Description:	The S3 access key belonging to the S3 key pair to remove.
Type:	String
Example:	`AB01C2D3EF45G6H7IJ8K`
Required:	Yes

`uid`

Description:	The user to remove the key from.
Type:	String
Example:	`foo_user`
Required:	No

`subuser`

Description:	The subuser to remove the key from.
Type:	String
Example:	`sub_foo`
Required:	No

`key-type`

Description:	Key type to be removed, options are: swift, s3. NOTE: Required to remove swift key.
Type:	String
Example:	`swift`
Required:	No

SPECIAL ERROR RESPONSES

None.

RESPONSE ENTITIES

None.

GET BUCKET INFO

Get information about a subset of the existing buckets. If `uid` is specified without `bucket` then all buckets beloning to the user will be returned. If `bucket` alone is specified, information for that particular bucket will be retrieved.

caps: buckets=read

SYNTAX

```
GET /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket to return info on.
Type:	String
Example:	`foo_bucket`
Required:	No

`uid`

Description:	The user to retrieve bucket information for.
Type:	String
Example:	`foo_user`
Required:	No

`stats`

Description:	Return bucket statistics.
Type:	Boolean
Example:	True [False]
Required:	No

RESPONSE ENTITIES

If successful the request returns a buckets container containing the desired bucket information.

`stats`

Description:	Per bucket information.
Type:	Container

`buckets`

Description:	Contains a list of one or more bucket containers.
Type:	Container

`bucket`

Description:	Container for single bucket information.
Type:	Container
Parent:	`buckets`

`name`

Description:	The name of the bucket.
Type:	String
Parent:	`bucket`

`pool`

Description:	The pool the bucket is stored in.
Type:	String
Parent:	`bucket`

`id`

Description:	The unique bucket id.
Type:	String
Parent:	`bucket`

`marker`

Description:	Internal bucket tag.
Type:	String
Parent:	`bucket`

`owner`

Description:	The user id of the bucket owner.
Type:	String
Parent:	`bucket`

`usage`

Description:	Storage usage information.
Type:	Container
Parent:	`bucket`

`index`

Description:	Status of bucket index.
Type:	String
Parent:	`bucket`

SPECIAL ERROR RESPONSES

`IndexRepairFailed`

Description:	Bucket index repair failed.
Code:	409 Conflict

CHECK BUCKET INDEX

Check the index of an existing bucket. NOTE: to check multipart object accounting with `check-objects`, `fix` must be set to True.

caps: buckets=write

SYNTAX

```
GET /{admin}/bucket?index&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket to return info on.
Type:	String
Example:	`foo_bucket`
Required:	Yes

`check-objects`

Description:	Check multipart object accounting.
Type:	Boolean
Example:	True [False]
Required:	No

`fix`

Description:	Also fix the bucket index when checking.
Type:	Boolean
Example:	False [False]
Required:	No

RESPONSE ENTITIES

`index`

Description:	Status of bucket index.
Type:	String

SPECIAL ERROR RESPONSES

`IndexRepairFailed`

Description:	Bucket index repair failed.
Code:	409 Conflict

REMOVE BUCKET

Delete an existing bucket.

caps: buckets=write

SYNTAX

```
DELETE /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket to remove.
Type:	String
Example:	`foo_bucket`
Required:	Yes

`purge-objects`

Description:	Remove a buckets objects before deletion.
Type:	Boolean
Example:	True [False]
Required:	No

RESPONSE ENTITIES

None.

SPECIAL ERROR RESPONSES

`BucketNotEmpty`

Description:	Attempted to delete non-empty bucket.
Code:	409 Conflict

`ObjectRemovalFailed`

Description:	Unable to remove objects.
Code:	409 Conflict

UNLINK BUCKET

Unlink a bucket from a specified user. Primarily useful for changing bucket ownership.

caps: buckets=write

SYNTAX

```
POST /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket to unlink.
Type:	String
Example:	`foo_bucket`
Required:	Yes

`uid`

Description:	The user ID to unlink the bucket from.
Type:	String
Example:	`foo_user`
Required:	Yes

RESPONSE ENTITIES

None.

SPECIAL ERROR RESPONSES

`BucketUnlinkFailed`

Description:	Unable to unlink bucket from specified user.
Code:	409 Conflict

LINK BUCKET

Link a bucket to a specified user, unlinking the bucket from any previous user.

caps: buckets=write

SYNTAX

```
PUT /{admin}/bucket?format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket name to unlink.
Type:	String
Example:	`foo_bucket`
Required:	Yes

`bucket-id`

Description:	The bucket id to unlink.
Type:	String
Example:	`dev.6607669.420`
Required:	Yes

`uid`

Description:	The user ID to link the bucket to.
Type:	String
Example:	`foo_user`
Required:	Yes

RESPONSE ENTITIES

`bucket`

Description:	Container for single bucket information.
Type:	Container

`name`

Description:	The name of the bucket.
Type:	String
Parent:	`bucket`

`pool`

Description:	The pool the bucket is stored in.
Type:	String
Parent:	`bucket`

`id`

Description:	The unique bucket id.
Type:	String
Parent:	`bucket`

`marker`

Description:	Internal bucket tag.
Type:	String
Parent:	`bucket`

`owner`

Description:	The user id of the bucket owner.
Type:	String
Parent:	`bucket`

`usage`

Description:	Storage usage information.
Type:	Container
Parent:	`bucket`

`index`

Description:	Status of bucket index.
Type:	String
Parent:	`bucket`

SPECIAL ERROR RESPONSES

`BucketUnlinkFailed`

Description:	Unable to unlink bucket from specified user.
Code:	409 Conflict

`BucketLinkFailed`

Description:	Unable to link bucket to specified user.
Code:	409 Conflict

REMOVE OBJECT

Remove an existing object. NOTE: Does not require owner to be non-suspended.

caps: buckets=write

SYNTAX

```
DELETE /{admin}/bucket?object&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket containing the object to be removed.
Type:	String
Example:	`foo_bucket`
Required:	Yes

`object`

Description:	The object to remove.
Type:	String
Example:	`foo.txt`
Required:	Yes

RESPONSE ENTITIES

None.

SPECIAL ERROR RESPONSES

`NoSuchObject`

Description:	Specified object does not exist.
Code:	404 Not Found

`ObjectRemovalFailed`

Description:	Unable to remove objects.
Code:	409 Conflict

GET BUCKET OR OBJECT POLICY

Read the policy of an object or bucket.

caps: buckets=read

SYNTAX

```
GET /{admin}/bucket?policy&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`bucket`

Description:	The bucket to read the policy from.
Type:	String
Example:	`foo_bucket`
Required:	Yes

`object`

Description:	The object to read the policy from.
Type:	String
Example:	`foo.txt`
Required:	No

RESPONSE ENTITIES

If successful, returns the object or bucket policy

`policy`

Description:	Access control policy.
Type:	Container

SPECIAL ERROR RESPONSES

`IncompleteBody`

Description:	Either bucket was not specified for a bucket policy request or bucket and object were not specified for an object policy request.
Code:	400 Bad Request

ADD A USER CAPABILITY

Add an administrative capability to a specified user.

caps:	users=write

SYNTAX

```
PUT /{admin}/user?caps&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID to add an administrative capability to.
Type:	String
Example:	`foo_user`
Required:	Yes

`user-caps`

Description:	The administrative capability to add to the user.
Type:	String
Example:	`usage=read,write;user=write`
Required:	Yes

RESPONSE ENTITIES

If successful, the response contains the user's capabilities.

`user`

Description:	A container for the user data information.
Type:	Container
Parent:	`user`

`user_id`

Description:	The user id.
Type:	String
Parent:	`user`

`caps`

Description:	User capabilities.
Type:	Container
Parent:	`user`

SPECIAL ERROR RESPONSES

`InvalidCapability`

Description:	Attempt to grant invalid admin capability.
Code:	400 Bad Request

EXAMPLE REQUEST

```
PUT /{admin}/user?caps&user-caps=usage=read,write;user=write&format=json HTTP/1.1
Host: {fqdn}
Content-Type: text/plain
Authorization: {your-authorization-token}
```

REMOVE A USER CAPABILITY

Remove an administrative capability from a specified user.

caps: users=write

SYNTAX

```
DELETE /{admin}/user?caps&format=json HTTP/1.1
Host {fqdn}
```

REQUEST PARAMETERS

`uid`

Description:	The user ID to remove an administrative capability from.
Type:	String
Example:	`foo_user`
Required:	Yes

`user-caps`

Description:	The administrative capabilities to remove from the user.

Type:	String
Example:	`usage=read, write`
Required:	Yes

RESPONSE ENTITIES

If successful, the response contains the user's capabilities.

`user`

Description:	A container for the user data information.
Type:	Container
Parent:	`user`

`user_id`

Description:	The user id.
Type:	String
Parent:	`user`

`caps`

Description:	User capabilities.
Type:	Container
Parent:	`user`

SPECIAL ERROR RESPONSES

`InvalidCapability`

Description:	Attempt to remove an invalid admin capability.
Code:	400 Bad Request

`NoSuchCap`

Description:	User does not possess specified capability.
Code:	404 Not Found

QUOTAS

The Admin Operations API enables you to set quotas on users and on bucket owned by users. See Quota Management for additional details. Quotas include the maximum number of objects in a bucket and the maximum storage size in megabytes.

To view quotas, the user must have a `users=read` capability. To set, modify or disable a quota, the user must have `users=write` capability. See the Admin Guide for details.

Valid parameters for quotas include:

- **Bucket:** The `bucket` option allows you to specify a quota for buckets owned by a user.
- **Maximum Objects:** The `max-objects` setting allows you to specify the maximum number of objects. A negative value disables this setting.
- **Maximum Size:** The `max-size` option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
- **Quota Type:** The `quota-type` option sets the scope for the quota. The options are `bucket` and `user`.
- **Enable/Disable Quota:** The `enabled` option specifies whether the quota should be enabled. The value should be either 'True' or 'False'.

GET USER QUOTA

To get a quota, the user must have `users` capability set with `read` permission.

```
GET /admin/user?quota&uid=<uid>&quota-type=user
```

SET USER QUOTA

To set a quota, the user must have `users` capability set with `write` permission.

```
PUT /admin/user?quota&uid=<uid>&quota-type=user
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

GET BUCKET QUOTA

To get a quota, the user must have `users` capability set with `read` permission.

```
GET /admin/user?quota&uid=<uid>&quota-type=bucket
```

SET BUCKET QUOTA

To set a quota, the user must have `users` capability set with `write` permission.

```
PUT /admin/user?quota&uid=<uid>&quota-type=bucket
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

SET QUOTA FOR AN INDIVIDUAL BUCKET

To set a quota, the user must have `buckets` capability set with `write` permission.

```
PUT /admin/bucket?quota&uid=<uid>&bucket=<bucket-name>&quota
```

The content must include a JSON representation of the quota settings as mentioned in Set Bucket Quota section above.

STANDARD ERROR RESPONSES

`AccessDenied`

Description:	Access denied.
Code:	403 Forbidden

`InternalError`

Description:	Internal server error.
Code:	500 Internal Server Error

`NoSuchUser`

Description:	User does not exist.
Code:	404 Not Found

`NoSuchBucket`

Description:	Bucket does not exist.
Code:	404 Not Found

`NoSuchKey`

Description:	No such access key.
Code:	404 Not Found

BINDING LIBRARIES

`Golang`

- QuentinPerez/go-radosgw

`Java`

- twonote/radosgw-admin4j

`Python`

- UMIACS/rgwadmin
- valerytschopp/python-radosgw-admin