ALOHA Load-Balancer

Web API Programmer's manual

Document version:	v1.0
API version concerned:	v2.0
Aloha version concerned:	v4.2.3
Last update date:	24th of November, 2011

RULE YOUR NETWORK!®

EXCELIANCE

[Sélectionnez la date]

Sommaire

WAPI Programmer's Manual

Version 1, November 2011

1. Introduction

This is a simple REST-like API based on the HTTP protocol. Objects are located in a hierarchical namespace and identified by an unique URI.

Each object can understand a subset of up to four HTTP methods: `GET`, `POST`, `PUT` and `DELETE`.

The API is composed of two distinct parts: a low layer, which is the command-line utility called `alohactl` [APICMD], and a high layer, which is an HTTP application called `wapi`.

The following schema illustrate how the layers are organized:

transport	nginx		
high layer	wapi		
low layer	alohactl		
components	haproxy	lvs	stunnel

2. Revision History

History and evolution of this API.

Version	Description	Status
1.0	First public preview	Deprecated
1.1	Updated meaning and usage of parameter values	Deprecated
2.0	Updated meaning and usage of parameter values	Supported

3. Structure

An **object** is identified by an URI in the namespace. Each **object** can be either an **file** or a **directory.**

The last element of an URI determine the name of the object.

A **file** is a list of key/value **parameters.**

A **directory** is a collection of **objects.**

In the following example, the object *myfarm1* returns a list of key/value parameters:

`GET`/api/2/trans/8L14RTM1kV/l7/farm/myfarm1

```
    {
        "key1": "value1",
        "key2": "value2",
        ...
    }
```

In the following example, the directory *farm* returns a list of objects:

`GET`/api/2/trans/8L14RTM1kV/l7/farm

```
    {
        "object1",
        "object2"
        ...
    }
```

4. Input/Output

Both input and output are always text, encoded with the **US-ASCII** character set.

4.1. Input

Two different input formats are available: HTTP query and JSON. They are equivalent and both can be used in most cases. However, JSON should be considered as the prefered format, since the HTTP query typing remains poor. It is possible that this format will disapear in a future evolution of the API. One advantage of using JSON is the ability to use the exact JSON output as an input, permitting convenient manipulation of objects.

4.1.1. HTTP query

HTTP query format is specified in RFC 2616 [RFC2616].
Input is formated as a list of `key=value` or `key`, separated by the `&` character.
When used without values, keys have to start with the prefix: `reset-`.
Example:

```
key=value&key&...
```

The following HTTP header field have to be set in the request:

```
Content-Type: text/plain
```

4.1.2. JSON

JSON format is specified in RFC 4627 [RFC4627].
Input is formated as a JSON object.
Keys are always `string` and **values** can be either `string`, `false` or `null`. The type `number` is currently ignored.
Example of every possible values:

```
{
    "key1": "value",
    "key5": null,
    ...
}
```

The following HTTP header field have to be set in the request:

```
Content-Type: application/json
```

4.1.3. Equivalence

Here is the equivalence between JSON and HTTP query formats:

JSON	HTTP query	alohactl
"key": "value"	key=value	--key value
"key": null	reset-key	--reset-key
"key": true		

The JSON value `true` is always ignored on input.
For example, the following JSON input:

```
{
    "protocol": "http",
    "log": "enabled",
    "log_format": "http",
    "client_inactivity_timeout": "25",
    "max_connections": "1000",
    "default_farm": "bk_myappli"
}
```
Is equivalent to the following HTTP query input:
```
    protocol=http&log=enabled&log_format=http&
    client_inactivity_timeout=25&max_connections=1000&
    default_farm=bk_myappli"
```
In the following sections, parameters will always be specified as JSON format.

4.2. Output

Output is always formatted as JSON, it can be either a single **object** or an **array** of string.

4.2.1. JSON array

Directories returns an array of names of the objects it contains.

Example:
```
    [
        "object1",
        "object2",
        ...
    ]
```
The following HTTP header field is set in the response:
```
    Content-Type: application/json
```

4.2.2. JSON object

Some files returns a JSON object.

The JSON object output format is identical to the JSON input format, in the way output from an object can be used directly as an input to another object of the same family.

However, an output of JSON object can also contain the value `true`.

For a description of the JSON object format, see section about JSON input.

4.2.3. Text

Some files returns plain text, encoded with US-ASCII character set.

The following HTTP header field is set in the response:
```
    Content-Type: text/plain
```

5. Special parameters

Some parameters have special meaning. They are spelt in uppercase letters. They are only used with few HTTP methods and can sometimes depend on another parameter.

Parameter	Value	Method	Depends on	Description
DEFAULT	a template	POST	nothing	specify a default template
METHOD	"clone"	POST	SOURCE	specify a special sub-method to call
SOURCE	an object	POST	METHOD=clone	specify an object to clone from

6. Transactions

Each request can be executed either **atomically** or as part of a **transaction**.
Both **atomic** requests and **transactions** affect only a particular **scope.**
An **atomic** request immediatly apply the changes on files.
A **transaction** must be **started** before issuing a serie of requests, then it can be either **committed** or **cancelled.**
Cancelling a transaction make no change on files and forgot the entiere serie of requests from the start of the transaction.
Committing a transaction consecutively apply the changes on files for the entiere serie of requests from the start of the transaction.

7. Request

7.1. URI

7.1.1. Version

All URIs start by the string `/api/2/`. Where `2` indicate the version of the API.
In case incompatible changes appears in the future, this format would permit to use different versions of the API concurrently.

7.1.2. Identication

The third element of URI should generally be either `scope` or `trans`, followed by the scope name or the transaction identifier.
For example:

```
/api/2/scope/bob/
```

or

```
/api/2/trans/KMQ6Z0VYsJ/
```

Where *bob* is a scope name and *KMQ6Z0VYsJ* is a transaction id.
`Trans` have to be used during a transaction and must refer to an existing transaction identifier, while `scope` have to be used an atomic request and can refer to any scope name.

7.2. Authentication

Every command require an HTTP Basic Authentication, as described in RFC 2616 [RFC2616].
An HTTP Basic Authentication appears as an HTTP header in the form:

```
Authorization: Basic YWRtaW46YWRtaW4=
```

Authentication is only permitted for the user `admin`, with its password specified in `/etc/passwd`.
An authentication failure returns the following HTTP headers:

```
Status: 401 Unauthorized
WWW-Authenticate: Basic realm="ALOHA"
```

7.2.1. Methods

Each object can understand up to four methods:

- o GET: display (return JSON object, JSON array of string or plain text),
- o POST: create (take JSON object),

o PUT: update (take JSON object),

o DELETE: delete.

When available, PUT accepts exactly the same format as POST but can accept a partial content. The PUT method only affect the specified parameters.

8. Objects and methods

Here is the list of all the objects and methods currently supported by the API.

As a convention, in the following sections, URIs wrote as `/api/2/*/*` mean both scope name or transaction id can be used as an identifier.

`/api/2/trans` `/api/2/scope/` <*name*> `/trans`

This documentation doesn't explain parameter meaning and usage. You have to refer to the *API Objects* documentation [APIOBJ] from the `alohactl` command-line utility.

8.1. Informations

`/api/2/version`

Method	Action	Result	Input	Output
GET	none	version number	none	text/plain

8.2. Transactions

`/api/2/trans`

Method	Action	Result	Input	Output
GET	none	list of transactions	none	application/json

`/api/2/scope/` <*name*> `/trans`

Method	Action	Result	Input	Output
GET	create a transaction	transaction id	none	text/plain

This object have an exceptional `GET` method behavior, which have a creation role.

`/api/2/trans/` <*id*>

Method	Action	Result	Input	Output
POST	commit a transaction	none	text/plain	none
DELETE	cancel a transaction	none	none	none

8.3. L7 farms and servers

8.3.1. Farms

8.3.1.1. Methods

`/api/2/*/*/l7/farm`

Method	Action	Result	Input	Output
GET	none	list of farms	none	application/json
DELETE	delete all farms	none	none	none

`/api/2/*/*/l7/farm/` <*name*>

Method	Action	Result	Input	Output
GET	none	show a farm	none	application/json
POST	create a farm	none	application/json text/plain	none
PUT	update a farm	none	application/json text/plain	none
DELETE	delete a farm	none	none	none

8.3.1.2. Parameters

The following parameters are handled by the methods POST and PUT.

Key
adv-check
adv-check-http-method
adv-check-http-uri
balance
check-fall
check-interval
check-port
check-rise
check-timeout
connect-failure-redispatch
connect-retries
connect-source
connect-timeout
connect-transparent
http-connection-mode
http-cookie
http-cookie-mode
http-cookie-nocache
http-pretend-keepalive
http-xff-header-insert
log
log-format
protocol
queued-timeout
server-inactivity-timeout

POST method can handle the special parameters `DEFAULT`, `METHOD=clone` and `SOURCE`.

8.3.2. Servers

8.3.2.1. Method

`/api/2/*/*/l7/farm/` *<name>* `/server`

Method	Action	Result	Input	Output
GET	none	list of servers	none	application/json
DELETE	delete all servers	none	none	none

`/api/2/*/*/l7/farm/` *<name>* `/server/` *<name>*

Method	Action	Result	Input	Output
GET	none	show a server	none	application/json
POST	create a server	none	application/json text/plain	none
PUT	update a server	none	application/json text/plain	none
DELETE	delete a server	none	none	none

8.3.2.2. Parameters

The following parameters are handled by the methods POST and PUT.

Key
address
check
http-cookie-id
maintenance
max-connections
port
sorry
weight

POST method can handle the special parameters `DEFAULT`, `METHOD=clone` and `SOURCE`.

8.4. L7 services and listeners

8.4.1. Services

8.4.1.1. Methods

`/api/2/*/*/l7/service`

Method	Action	Result	Input	Output
GET	none	list of services	none	application/json
DELETE	delete all services	none	none	none

`/api/2/*/*/l7/service/` <*name*>

Method	Action	Result	Input	Output
GET	none	show a service	none	application/json
POST	create a service	none	application/json text/plain	none
PUT	update a service	none	application/json text/plain	none
DELETE	delete a service	none	none	none

8.4.1.2. Parameters

The following parameters are handled by the methods POST and PUT.

Key
client-inactivity-timeout
default_farm
http-connection-mode
http-keepalive-timeout
http-pretend-keepalive
http-request-timeout
log
log-format
log-ignore-null
max-connections
protocol

POST method can handle the special parameters `DEFAULT`, `METHOD=clone` and `SOURCE`.

8.4.2. Listeners

8.4.2.1. Methods

`/api/2/*/*/l7/service/` *<name>* `/listener`

Method	Action	Result	Input	Output
GET	none	list of listeners	none	application/json
DELETE	delete all listeners	none	none	none

`/api/2/*/*/l7/service/` *<name>* `/listener/` *<name>*

Method	Action	Result	Input	Output
GET	none	show a listener	none	application/json
POST	create a listener	none	application/json text/plain	none
PUT	update a listener	none	application/json text/plain	none
DELETE	delete a listener	none	none	none

8.4.2.2. Parameters

The following parameters are handled by the methods POST and PUT.

Key
address
port
ssl
transparent

POST method can handle the special parameters `DEFAULT`, `METHOD=clone` and `SOURCE`.

8.5. L4 farms and servers

8.5.1. Farms

8.5.1.1. Methods

`/api/2/*/*/l4/farm`

Method	Action	Result	Input	Output
GET	none	list of farms	none	application/json
DELETE	delete all farms	none	none	none

`/api/2/*/*/l4/farm/` *<name>*

Method	Action	Result	Input	Output
GET	none	show a farm	none	application/json
POST	create a farm	none	application/json text/plain	none
PUT	update a farm	none	application/json text/plain	none
DELETE	delete a farm	none	none	none

8.5.1.2. Parameters

The following parameters are handled by the methods POST and PUT.

Key
adv-check-http-status-code
adv-check-http-uri
adv-check
balance
check-interval
check-port
check-source
check-timeout
mode
persistence
service-address
service-port
service-protocol

POST method can handle the special parameters `DEFAULT`, `METHOD=clone` and `SOURCE`.

8.5.2. Servers

8.5.2.1. Methods

`/api/2/*/*/l4/farm/` *<name>* `/server`

Method	Action	Result	Input	Output
GET	none	list of servers	none	application/json
DELETE	delete all servers	none	none	none

`/api/2/*/*/l4/farm/` *<name>* `/server/` *<name>*

Method	Action	Result	Input	Output
GET	none	show a server	none	application/json
POST	create a server	none	application/json text/plain	none
PUT	update a server	none	application/json text/plain	none
DELETE	delete a server	none	none	none

8.5.2.2. Parameters

The following parameters are handled by the methods POST and PUT.

Key
address
check
sorry
port
weight

POST method can handle the special parameters `DEFAULT`, `METHOD=clone` and `SOURCE`.

8.6. System

8.6.1. Local

`/api/2/sys/local/save`

Method	Action	Result	Input	Output
POST	save configuration	none	text/plain	none

8.6.2. Peers

`/api/2/sys/peers/0/save`

Method	Action	Result	Input	Output
POST	save configuration on peer	none	text/plain	none

`/api/2/sys/peers/0/push`

Method	Action	Result	Input	Output
POST	push configuration to peer	none	text/plain	none

9. Error handling

9.1. Success

All commands returns the following HTTP header field in case of success:

```
Status: 200 OK
```

9.2. Errors

Commands should set the HTTP header field `Status` on error, and optionnaly the following HTTP header fields:

```
X-Alctl-Errno
X-Alctl-Errstr
```

Three different type of error can happen:

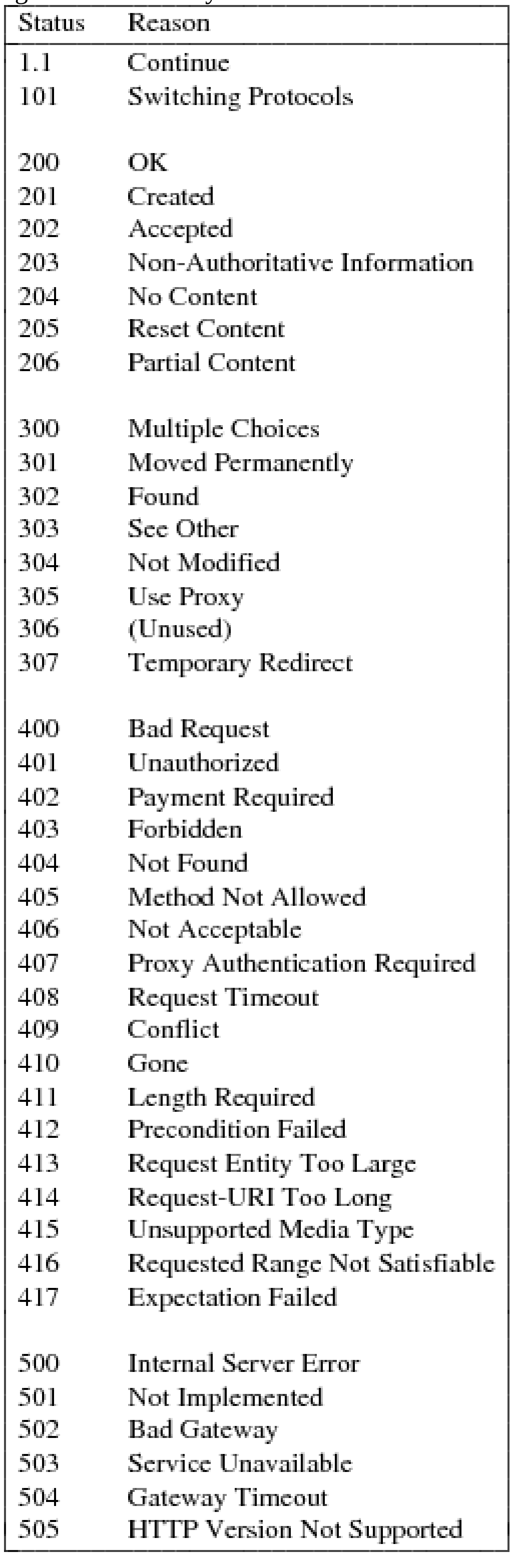
o errors returned by the HTTP server, returning `Status`.

o errors returned by the high-layer API, returning `Status` and optionnaly `X-Alctl-Errno` and `X-Alctl-Errstr`.

o errors returned by the low-layer API, returning `Status`, `X-Alctl-Errno` and `X-Alctl-Errstr`.

9.3. HTTP Status

These status and reason messages are returned by the HTTP server.



Status	Reason
1.1	Continue
101	Switching Protocols
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
306	(Unused)
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

9.4. High-layer errors

There errors are returned by the high-layer of the API.

Status	Reason	Usual cause
400	Bad Request	client error, see lower
404	Not Found	this object doesn't exist
405	Method Not Allowed	this method doesn't exist for this object
415	Unsupported Media Type	this file type isn't supported by thus object
500	Internal Server Error	server error, see lower

HTTP errors 400 and 500 are completed by the header fields `X-Alctl-Errno` and `X-Alctl-Errstr`.

Status	Reason	X-Alctl-Errno	X-Alctl-Errstr
500	Internal Server Error	1001	should not happen
500	Internal Server Error	1011	read error
500	Internal Server Error	1012	write error
500	Internal Server Error	1013	execution error
500	Internal Server Error	1021	lckpwdf failed
400	Bad Request	1101	missing scope name
400	Bad Request	1102	missing transaction id
400	Bad Request	1103	missing scope name or transaction id
400	Bad Request	1111	bad input format
400	Bad Request	1112	empty input
400	Bad Request	1113	too much input

The `X-Alctl-Errstr` header field string is only available for information purpose and the result may differ. You should only consider `X-Alctl-Errno`.

These errors follows a categorization rule:

Type	X-Alctl-Errno	Category
Server error	10xx	
	100x	general
	101x	file system
	102x	locking
Client error	11xx	
	110x	identication
	111x	input format

9.5. Low-layer errors

These errors are returned by the low-layer of the API, ie the `alohactl` command-line utility.

Status	Reason	X-Alctl-Errno	X-Alctl-Errstr
400	Bad Request	0	...
400	Bad Request	99	...
503	Service Unavailable	100	API is locked
500	Internal Server Error	101	Configuration validation failure
500	Internal Server Error	110	Configuration apply failure
500	Internal Server Error	111	Unable to create transaction context
500	Internal Server Error	112	Unable to create transaction context for sub-module
500	Internal Server Error	113	Unable to re-create transaction context
500	Internal Server Error	114	Unable to backup configuration
500	Internal Server Error	115	Unable to install config
500	Internal Server Error	120	Unable to restore config

The `X-Alctl-Errstr` header field string is only available for information purpose and the result may differ. You should only consider `X-Alctl-Errno.`

10. Examples

In the following examples, we will consider that you set the variable `ADDR` to the address of your server.

Example:

```
ADDR=192.168.0.1.1:4444
```

Start a new transaction for scope *bob:*

```
$ curl -i --user admin:admin
http://$ADDR/api/2/scope/bob/trans
HTTP/1.1 200 OK
Server: ALOHA/WUI
Date: Mon, 10 Oct 2011 12:00:00 GMT
Content-Type: text/plain; charset=us-ascii
Transfer-Encoding: chunked
Connection: keep-alive
Cache-control: no-cache
8L14RTM1kV
```

Create a farm *myfarm1* in transaction *8L14RTM1kV:*

```
$ cat << EOF > output
{
{
    "balance": "roundrobin",
    "protocol": "http",
    "log": "enabled",
    "log_format": "http",
    "http_xff_header_insert": "enabled",
    "http_cookie": "enabled",
    "http_cookie_name": "SERVERID",
    "http_cookie_mode": "set-silent",
    "http_cookie_nocache": "enabled",
    "check_interval": "3",
    "check_rise": "2",
    "check_fall": "3",
    "adv_check": "http",
    "adv_check_http_method": "HEAD",
    "adv_check_http_uri": "/",
    "server_inactivity_timeout": "25"
}
}
EOF
$ curl -i --user admin:admin
-d @output -H ’Content-Type: application/json’
http://$ADDR/api/2/trans/8L14RTM1kV/l7/farm/myfarm1
HTTP/1.1 200 OK
Server: ALOHA/WUI
Date: Mon, 10 Oct 2011 12:00:00 GMT
Content-Type: text/plain; charset=us-ascii
Transfer-Encoding: chunked
Connection: keep-alive
Cache-control: no-cache
```

Show the farm *myfarm1* in transaction *8L14RTM1kV:*

```
    $ curl -i --user admin:admin
    http://$ADDR/api/2/trans/8L14RTM1kV/l7/farm/myfarm1
    HTTP/1.1 200 OK
    Server: ALOHA/WUI
    Date: Mon, 10 Oct 2011 12:00:00 GMT
    Content-Type: application/json; charset=us-ascii
    Transfer-Encoding: chunked
    Connection: keep-alive
    Cache-control: no-cache
    {
        "balance": "roundrobin",
        "protocol": "http",
        "log": "enabled",
        "log_format": "http",
        "http_connection_mode": null,
        "http_pretend_keepalive": null,
        "http_xff_header_insert": "enabled",
        "http_cookie": "enabled",
        "http_cookie_name": "SERVERID",
        "http_cookie_mode": "set-silent",
        "http_cookie_nocache": "enabled",
        "check_interval": "3",
        "check_rise": "2",
        "check_fall": "3",
        "check_port": null,
        "check_timeout": null,
        "adv_check": "http",
        "adv_check_http_method": "HEAD",
        "adv_check_http_uri": "/",
        "queued_timeout": null,
        "connect_timeout": null,
        "connect_retries": null,
        "connect_failure_redispatch": null,
        "connect_source": null,
        "connect_transparent": null,
        "server_inactivity_timeout": "25"
    }
```

Create a server *myserver1* in farm *myfarm1:*

```
    $ cat << EOF > output
    {
        "address": "192.168.1.1",
        "port": "80",
        "max_connections": "1000",
        "weight": "10",
        "http_cookie_id": "s1",
        "check": "enabled"
    }
    EOF
    $ curl -i --user admin:admin
    -d @output -H 'Content-Type: application/json'
    http://$ADDR/api/2/trans/8L14RTM1kV/l7/farm/myfarm1/server/myserver
    1
    HTTP/1.1 200 OK
    Server: ALOHA/WUI
    Date: Mon, 10 Oct 2011 12:00:00 GMT
    Content-Type: text/plain; charset=us-ascii
    Transfer-Encoding: chunked
    Connection: keep-alive
    Cache-control: no-cache
```

Update the address to *192.168.1.2.* of the server *myserver1:*

```
    $ cat << EOF > output
    {
        "address": "192.168.1.2"
    }
    EOF
    $ curl -i --user admin:admin
    -T output -H 'Content-Type: application/json'
    http://$ADDR/api/2/trans/8L14RTM1kV/l7/farm/myfarm1/server/myserver
    1
    HTTP/1.1 200 OK
    Server: ALOHA/WUI
    Date: Mon, 10 Oct 2011 12:00:00 GMT
    Content-Type: text/plain; charset=us-ascii
    Transfer-Encoding: chunked
    Connection: keep-alive
    Cache-control: no-cache
```

Show the server *myserver1* in the farm *myfarm1:*

```
    $ curl -i --user admin:admin
```

```
http://$ADDR/api/2/trans/8L14RTM1kV/l7/farm/myfarm1/server/myserver
1
HTTP/1.1 200 OK
Server: ALOHA/WUI
Date: Mon, 10 Oct 2011 12:00:00 GMT
Content-Type: application/json; charset=us-ascii
Transfer-Encoding: chunked
Connection: keep-alive
Cache-control: no-cache
{
    "address": "192.168.1.2",
    "port": "80",
    "max_connections": "1000",
    "weight": "10",
    "http_cookie_id": "s1",
    "sorry": null,
    "check": "enabled",
    "maintenance": null
}
```

Commit transaction *8L14RTM1kV:*

```
$ > empty
$ curl -i --user admin:admin
-d @empty -H ’Content-Type: text/plain’
http://$ADDR/api/2/trans/8L14RTM1kV
HTTP/1.1 200 OK
Server: ALOHA/WUI
Date: Fri, 01 Jul 2011 12:06:00 GMT
Content-Type: text/plain; charset=us-ascii
Transfer-Encoding: chunked
Connection: keep-alive
Cache-control: no-cache
ALCTL Notice: Commit on scope: bob
```

Clone the farm *myfarm1* to the farm *myfarm2:*

```
$ cat << EOF > output
{
    "METHOD": "clone",
    "SOURCE": "myserver1"
}
EOF
$ curl -i --user admin:admin
-d  @output -H ’Content-Type: application/json’
http://$ADDR/api/2/scope/bob/l7/farm/myfarm1/server/myserver2
HTTP/1.1 200 OK
Server: ALOHA/WUI
Date: Fri, 01 Jul 2011 12:07:00 GMT
Content-Type: text/plain; charset=us-ascii
Transfer-Encoding: chunked
Connection: keep-alive
```

```
Cache-control: no-cache
```

Cancel transaction *VUJ94GGIJj:*

```
curl -i --user admin:admin
-X DELETE
http://$ADDR/api/2/trans/VUJ94GGIJj
HTTP/1.1 200 OK
Server: ALOHA/WUI
Date: Fri, 01 Jul 2011 12:08:00 GMT
Content-Type: text/plain; charset=us-ascii
Transfer-Encoding: chunked
Connection: keep-alive
Cache-control: no-cache
```

References

[RFC2616] ‘‘Hypertext Transfer Protocol -- HTTP/1.1’’

[RFC4627] ‘‘The application/json Media Type for JavaScript Object Notation (JSON)’’

[APICMD] ‘‘alohactl commands documentation’’, `alohactl-commands-doc.txt`

[APIOBJ] ‘‘API Objects documentation’’, `api-objects-doc.txt`