



ZFS STORAGE
APPLIANCE

An Oracle Technical White Paper

September 2014

Working with the RESTful API for the Oracle ZFS Storage Appliance

Table of Contents

Introduction	3
RESTful API Architecture in the Oracle ZFS Storage Appliance	4
Success and Error Return Codes	5
Simple Examples	6
Authentication and Sessions	8
REST Service Versions	8
Using Integrated Development Environments	9
Program Examples	10
Using <code>curl</code> in Shell Scripts	10
Using Python	12
Python programming best practices	13
Python code examples	13
Conclusion	20
References	21
Appendix A: Python Code for <code>restmulty.py</code> Module	22
Appendix B: Python Code for <code>restclient.py</code> Module	25

Introduction

The Oracle ZFS Storage Appliance combines advanced hardware and software architecture for a multiprotocol storage subsystem that enables users to simultaneously run a variety of application workloads and offer advanced data services. First-class performance characteristics are illustrated by the results of the industry standard benchmarks like SPC-1, SPC-2 and SPECsfs.

The Oracle ZFS Storage Appliance provides an Application Programming Interface (API) based on the Representational State Transfer (REST) architectural style. REST is designed to provide a consistent interface to the roles of components, their functional interactions and state data while hiding the specific implementation and protocol syntax details for a particular application or system.

REST is an industry standard developed by the W3C Technical Architecture Group – based on HTTP 1.1. A REST API is known as RESTful as it adheres to the REST constraints which are detailed in "Architectural Styles and the design of Network-based Software Architectures," the Doctoral dissertation by Roy Fielding at the University of California, Irvine, in 2000.

There are only four REST methods – GET, PUT, POST, DELETE. With the obvious exception of the DELETE method, these methods are those that are used by web browsers to access web sites. These methods are also described as CRUD – Create, Read, Update and Delete – operations.

For the Oracle ZFS Storage Appliance, REST is designed for use in connecting systems management monitoring and control software to allow automated and manual control and monitoring of the components and services with the Oracle ZFS Storage Appliance without using either the command line interface (CLI) or direct browser user interface (BUI). REST can also be used for iterative tasks in a programming environment such as Python. In this sense, REST is not a storage protocol but an administrative interface.

RESTful API Architecture in the Oracle ZFS Storage Appliance

The RESTful API supplements the access client methods offered by the Oracle ZFS Storage Appliance family of products. The three supported client types are:

- CLI: SSH - Login - session
- BUI: HTTP - HTML/XML - Cookie based session
- REST: HTTP - JSON – Sessionless

The following graphic illustrates the client types and their architecture within the Oracle ZFS Storage Appliance.

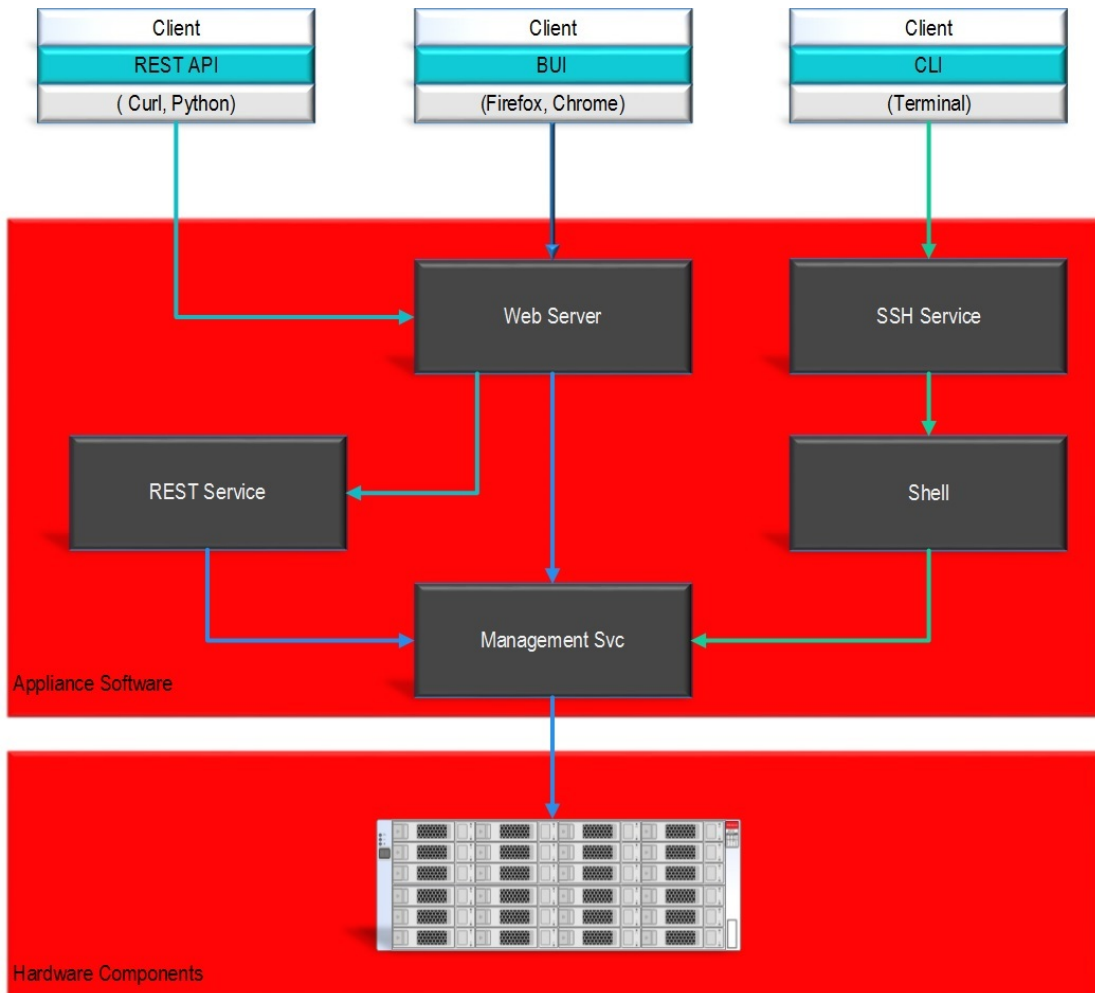


Figure 1. Client architecture for communicating with the Oracle ZFS Storage Appliance

The REST service supports any HTTP client conforming to HTTP 1.0 or HTTP 1.1.

Previously, operations were carried out on the Oracle ZFS Storage Appliance using SSH as the transport mechanism. The utility of this setup was hampered by the inability to return the status of the operation without some interpretive wrapper around the command execution.

With the advent of REST within the Oracle ZFS Storage Appliance, success or failure of the command is returned in parsable JavaScript Object Notation (JSON) format. This means that large jobs with similar operations can be carried out with proper error detection and, if necessary, remedial action also initiated by a comprehensive script.

One example where this may be useful is in the creation and masking of many LUNs in a virtual desktop infrastructure (VDI) environment. Typically this involves similar operations being carried out with small variations in the masking details and naming of LUNs. Written in any of the supported scripting languages, this tedious task can now be carried out with relative ease and with full error reporting, so that any problems are caught and dealt with as early as possible.

Access to the RESTful API is through the standard HTTPS interface:

<https://zfssa.example.com:215/api>

The following figure and table represent and detail the operations the REST service offers.

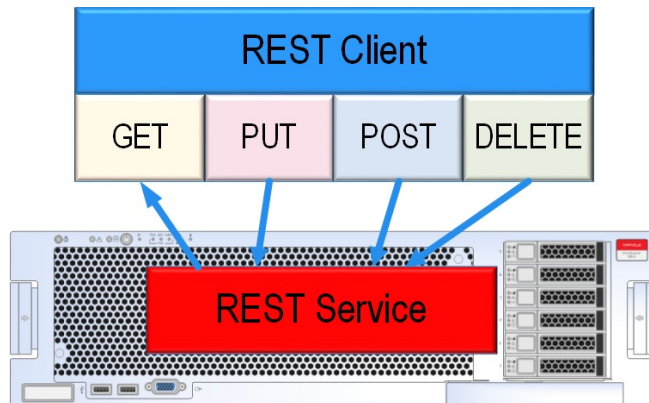


Figure 2. The REST Service operations

TABLE 1. CRUD OPERATIONS	
OPERATION	USE
GET	List information about a resource – for example, storage pools, projects, LUNs, shares, users, and so on
POST	Create a new resource – <code>POST /storage/v1/pools</code> creates a new pool, for example
PUT	Modify a resource
DELETE	Destroy a resource

Success and Error Return Codes

The response body from the API is encoded in JSON format (RFC 4627.) Unless otherwise stated, a single resource returns a single JSON result object with the resource name as a property. Similarly, unless otherwise stated, the create (POST) and modify (PUT) commands return the properties of the appropriate resource.

Errors return an HTTP status code indicating the error, along with the fault response payload which is formatted like the following:

```
{
  fault: {
    message: 'ERR_INVALID_ARG',
    details: 'Error Details',
    code: 500
  }
}
```

Successful requests will return one of four codes, depending on context:

TABLE 2. SUCCESS RETURN CODES		
NAME	CODE	DESCRIPTION
OK	200	Request returned success
CREATED	201	New resource created successfully
ACCEPTED	202	The request was accepted
NO_CONTENT	204	Command returned OK but no data will be returned

The following table defines some common error codes:

TABLE 3. ERROR RETURN CODES		
NAME	CODE	DESCRIPTION
ERR_INVALID_ARG	400	Invalid input argument
ERR_UNKNOWN_ARG	400	Extra unhandled input argument
ERR_MISSING_ARG	400	Required input argument missing
ERR_UNAUTHORIZED	401	The user is not authorized to execute command
ERR_DENIED	403	Operation denied
ERR_NOT_FOUND	404	The requested item was not found
ERR_OBJECT_EXISTS	409	Request created an object that already exists
ERR_OVER_LIMIT	413	Input request too large to handle
ERR_UNSUPPORTED_MEDIA	415	Requested media type is not supported by request
ERR_NOT_IMPLEMENTED	501	Operation not implemented
ERR_BUSY	503	Service not available due to limited resources

Simple Examples

The following example shows the RESTful API in use. This Python script uses the GET operation to download entries in the audit log files:

```
from restclientlib import *
host = "10.0.2.13"
user = "root"
password = "secret"

client = RestClient (host)
result = client.login (user, password)

result = client.get("/api/log/v1/collect/audit")
print result.getdata()
client.logout()
```

Assuming the username, password and host are correctly set, the following output results from running the script:

```
Thu Apr 17 13:08:16 2014
nvlist version: 0
address = 10.0.2.15
host = 10.0.2.15
annotation =
user = root
class = audit.ak.xmlrpc.system.login_success
payload = (embedded nvlist)
nvlist version: 0
iscli = 0
(end payload)

summary = User logged in

Thu Apr 17 12:10:32 2014
nvlist version: 0
address = 10.0.2.15
host = 10.0.2.15
annotation =
user = root
class = audit.ak.appliance.nas.storage.configure
payload = (embedded nvlist)
nvlist version: 0
pool = onlystuff
```

```

profile = Striped
(end payload)

summary = Configured storage pool "onlystuff" using profile "Striped"

Thu Apr 17 12:11:04 2014
nvlist version: 0
address = 10.0.2.15
host = 10.0.2.15
annotation =
user = root
class = audit.ak.xmlrpc.svc.enable
payload = (embedded nvlist)
nvlist version: 0
service = rest
(end payload)

summary = Enabled rest service

Thu Apr 17 12:24:01 2014
nvlist version: 0
address = 10.0.2.15
host = 10.0.2.15
annotation =
user = root
class = audit.ak.xmlrpc.system.session_timeout
payload = (embedded nvlist)
nvlist version: 0
iscli = 0
(end payload)

summary = Browser session timed out

Thu Apr 17 13:10:28 2014
nvlist version: 0
host = <console>
annotation =
user = root
class = audit.ak.xmlrpc.system.logout
payload = (embedded nvlist)
nvlist version: 0
iscli = 1
(end payload)

summary = User logged out of CLI
...

```

Another example creates multiple shares (in this case, 10) in a given pool and project:

```

#!/usr/bin/python

from restclientlib import *

host = "10.0.2.13"
user = "root"
password = "secret"

pool="R1Pool"
project = "apiproj"

sharepath = "/api/storage/v1/pools/%s/projects/%s/filesystems"

client = RestClient(host)
result = client.login(user, password)
for i in range(1, 10+1):
    sharename="MyShare_%d" % i
    result=client.post(sharepath % (pool, project), { "name": sharename })
    if result.status != httplib.CREATED:
        print result.status
        print "Error creating " + sharename + ": " + result.body
client.logout()

```

In this last example, the errors in creating the shares are tracked but the loop continues regardless.

More complex examples are presented in a following section.

Authentication and Sessions

The REST service uses the same underlying user authentication as the Oracle ZFS Storage Appliance BUI and CLI services.

Authentication can take one of two forms: Basic or User. Basic authentication requires that each request contain a valid username and password while User authentication requires that the X-Auth-User header contain the username and the X-Auth-Key contain the password.

Once a session has been successfully authenticated through either method, a session header is returned and can subsequently be used for future requests until the session expires, at which point re-authentication must take place.

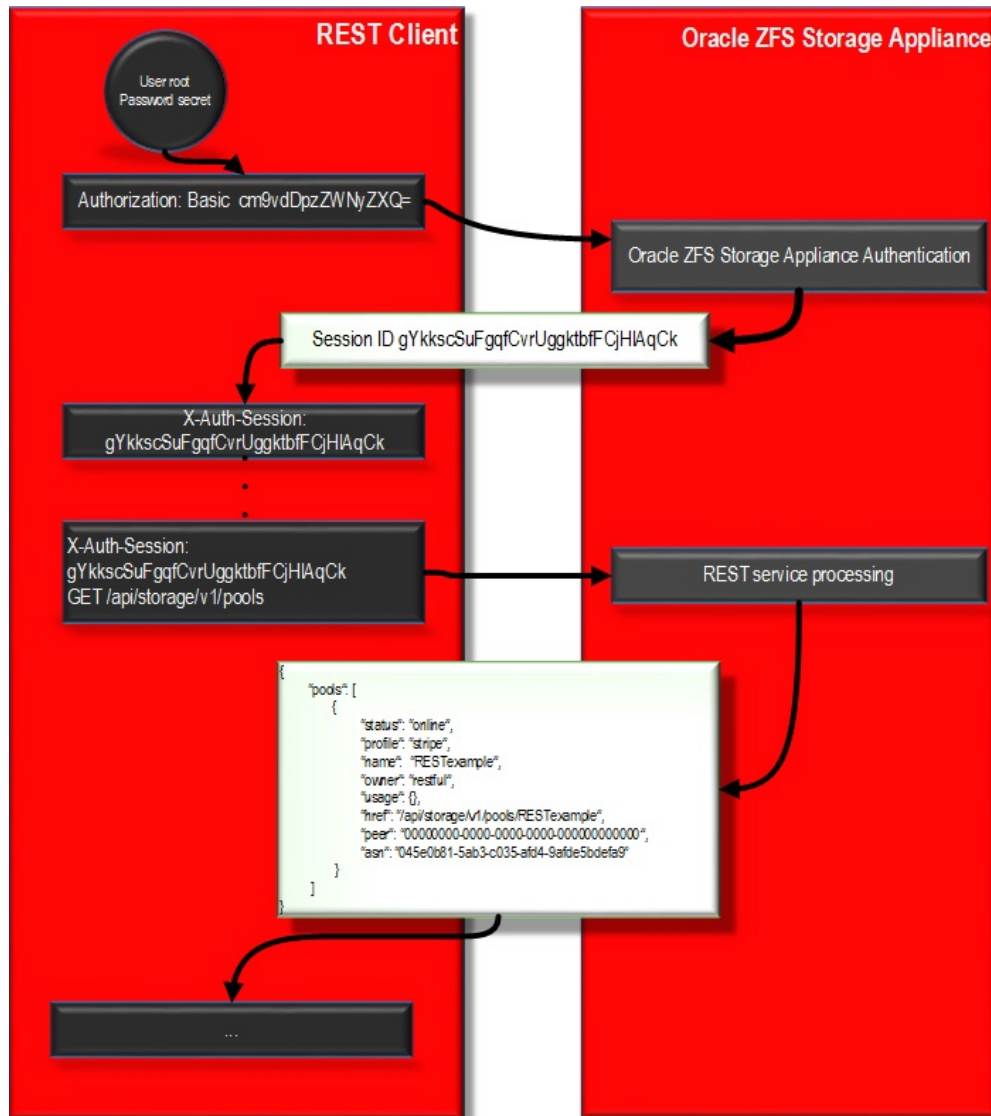


Figure 3. Session variable use

REST Service Versions

Each service has a version number embedded as part of the Uniform Resource Identifier (URI) to access the REST service. For example: `/api/user/v1/users`

The version numbering consists of a major and minor revision. While the major version number must be supplied, the minor is optional and defaults to '0'.

The major number must match the major number of the Oracle ZFS Storage Appliance RESTful API software. The minor number, should it be supplied, must be less than or equal to the minor number of the RESTful API service.

The following table shows the results of requests to a service which is running version 2.1 of the RESTful API software.

TABLE 4. SUCCESS RETURN CODES	
REQUEST VERSION	RESULT
v1	ERROR – major number does not match
v2	Success – Major number matches and implied minor '0' is less than or equal to minor version 1.
v2.1	Success – Major and minor numbers both match
v2.2	ERROR – Major matches but minor is greater than the service version

Using Integrated Development Environments

There are three areas where the Oracle ZFS Storage Appliance RESTful API can be used to externally manage an Oracle ZFS Storage Appliance:

- Using scripts to execute repetitive tasks, like creating a large number of shares
- Creating scripts/programs with specific tasks for administrators
- Integrating a customer monitoring and management environment, like the OpenStack environment, with the Oracle ZFS Storage Appliance

Each of these options requires some coding development to implement the required user/administrator functionality. Several programming languages can be used for this. The choice of language depends on the programming rules and standards enforced in a customer environment. Sometimes regulatory requirements influence the choice of program language. Python, Ruby, PHP and Java are a few of the most popular choices.

A key requirement is the support for JavaScript Object Notation (JSON) in the programming environment of choice. It is a lightweight data interchange format used by the RESTful API to exchange data between the client and the Oracle ZFS Storage Appliance.

The simplest way to write code is to use a text editor, write code, and run it through the language interpreter program or compile it to create a direct executable program. Test and debug the program and update the code source with the text editor. This works fine for simple scripts and/or programs. When the number of lines of code increases from just a few lines to multiple modules, using an Integrated Development Environment (IDE) makes more sense.

IDEs consists of a combined code text editor and a code compilation/debug environment. The text editor often has extra features to format text according to general accepted coding standards and checks for coding syntax errors. This enhances the quality of the code and helps to enforce a uniform way of writing code text within an engineering group.

This document reflects Python as the coding language and the free Community Edition of PyCharm as the IDE. The following figure shows a typical PyCharm setup, using a navigation pane on the left,

showing the various Python modules used for the current project, a code editor on the top right, and a debugger/console pane at the bottom.

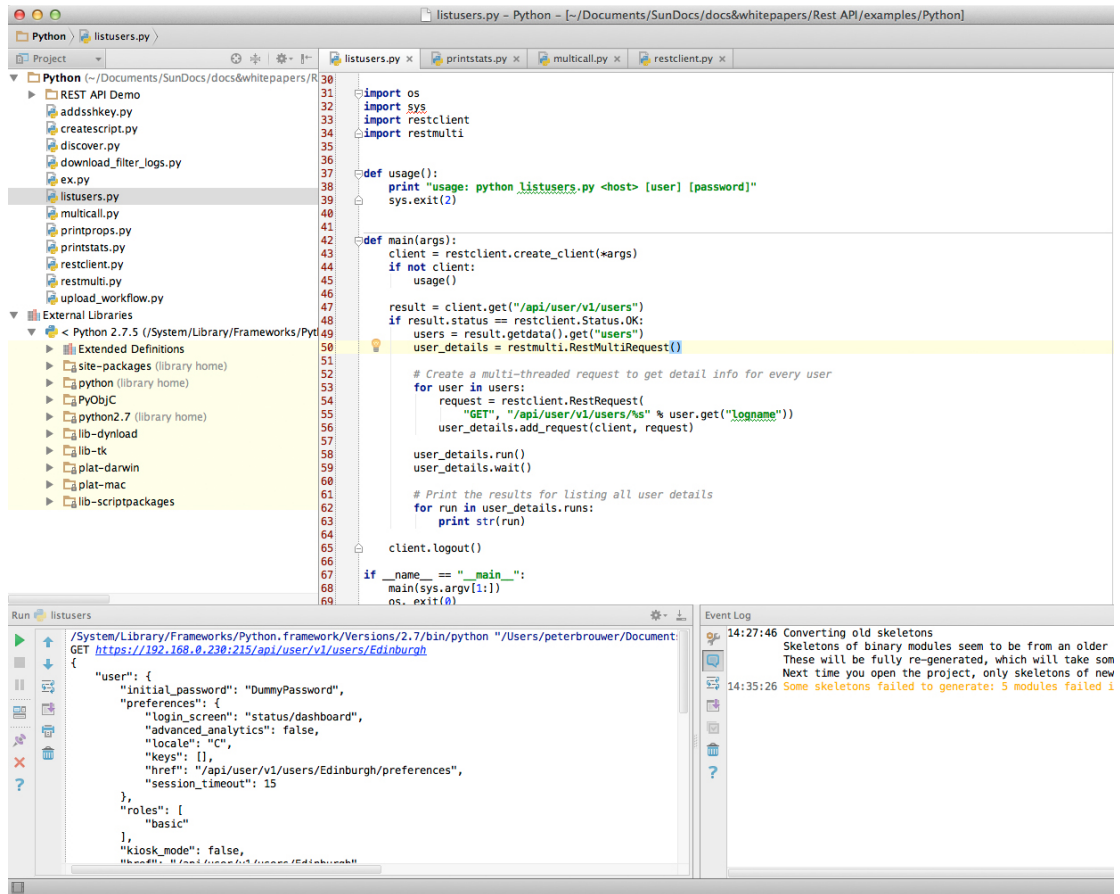


Figure 4. PyCharm IDE screen view

Program Examples

Regardless of the programming environments used for the RESTful API, the principle remains the same: communication between the client program and the Oracle ZFS Storage Appliance is based on simple HTTP use. The following examples illustrate the use of the RESTful API using the CURL utility in a shell scripting CLI-type environment and a Python programming environment. The examples illustrate the use of the API commands. Error handling is rudimentary.

Using curl in Shell Scripts

The following example shows a framework for using `curl` in a shell script to execute the GET, PUT, POST and DELETE commands through `curl`. The URL path of the resource to operate on has to be provided as argument for the script. User login credentials can either be specified using the `-u` and `-p` argument options or set using the environment variables `$USER` and `$PASSWORD`.

```

1  #!/bin/bash
2  #
3  # Example 1
4  # Copyright (c) 2013, 2014 Oracle and/or its affiliates. All rights reserved.
5  # Script akrest
6
7  CURL=(`which curl` -3 -k)          # curl command options
8  ACCEPT="application/json"         # Default returned content type accepted
9  DO_FORMAT=false                   # Pretty print JSON output
10 PYTHON=`which python`             # Used for pretty printing JSON output
11 USER=$ZFSSA_USER                  # Login user

```

```

12 PASSWORD=${ZFSSA_PASSWORD}           # Login password
13 SESSION=${ZFSSA_SESSION}             # Login session id
14 INFILE=                               # POST/PUT input file
15 CONTENT="application/json"           # Default input content type
16 VERBOSE=false                         # Print more data
17
18 usage() {
19     echo "usage akrest [options] <host> <get|post|put|delete> <path> [json]"
20     echo "options:"
21     echo "    -f Format output"
22     echo "    -h Print headers"
23     echo "    -c Request CLI script"
24     echo "    -i <file> Input file to post/put"
25     echo "    -s <id> Session id"
26     echo "    -p <pass> Login password"
27     echo "    -u <user> Login username"
28     echo "    -v Verbose"
29     echo "    -y Request YAML output"
30     echo "    -z Request compressed return data (only some commands supported)"
31     exit 2
32 }
33
34 while getopts u:p:i:s:hvbcfyz name
35 do
36     case $name in
37         c) CURL=( "${CURL[@]}" "--header" "X-Zfssa-Get-Script: true" );;
38         b) CONTENT="application/octet-stream";;
39         f) DO_FORMAT="true";;
40         u) USER="$OPTARG";;
41         p) PASSWORD="$OPTARG"
42           SESSION=;;
43         h) CURL=( "${CURL[@]}" "-i" );;
44         i) INFILE=$OPTARG;;
45         s) CURL=( "${CURL[@]}" "--header" "X-Auth-Session: $OPTARG" )
46           PASSWORD="";;
47         v) VERBOSE="true"
48           CURL=( "${CURL[@]}" "-v" );;
49         y) ACCEPT="text/x-yaml";;
50         z) CURL=( "${CURL[@]}" "--header" "Accept-Encoding: gzip" );;
51         ?) usage
52     esac
53 done
54 shift $(( $OPTIND - 1 ))
55
56 if [ "$#" == "3" ]; then
57     JSON=""
58 elif [ "$#" == "4" ]; then
59     JSON=$4
60     CURL=( "${CURL[@]}" "-d" "@-" "--header" "Content-Type: ${CONTENT}" )
61 else
62     usage
63 fi
64
65 HOST=$1
66 REQUEST=$2
67 PATH=$3
68 DATA=$4
69
70 case $REQUEST in
71     get) REQUEST=GET;;
72     put) REQUEST=PUT;;
73     post) REQUEST=POST;;
74     delete) REQUEST=DELETE;;
75     *) usage
76 esac
77
78 if [ "$HOST" == "" ]; then
79     usage
80 fi
81 if [ "$PATH" == "" ]; then
82     usage
83 fi
84 if [ "localhost" == "$HOST" ]; then
85     URL="http://$HOST:8215/$PATH"
86 else
87     URL="https://$HOST:215/api/$PATH"
88 fi
89
90 if [ "${USER}" == "" ]; then
91     USER=root
92 fi
93 if [ "${SESSION}" != "" ]; then
94     CURL=( "${CURL[@]}" "--header" "X-Auth-Session: ${SESSION}" )
95 elif [ "${PASSWORD}" != "" ]; then
96     CURL=( "${CURL[@]}" "--user" "${USER}:${PASSWORD}" )
97 else
98     if [ "$HOST" != "localhost" ]; then

```

```

99         echo "Either password or session needs to be set"
100         exit 1
101     fi
102 fi
103
104 if [ "${INFILE}" == "" ]; then
105     CURL=( "${CURL[@]}" "-sS" )
106 else
107     CURL=( "${CURL[@]}" "-d" "@${INFILE}" "--header" "Content-Type: $CONTENT" )
108 fi
109
110 CURL=( "${CURL[@]}" "--header" "Accept: ${ACCEPT}" -X "${REQUEST}" "${URL}" )
111
112 if [ "${VERBOSE}" == "true" ]; then
113     echo "${CURL[@]}"
114 fi
115
116 if [ "${DO_FORMAT}" == "true" ]; then
117     if [ "$JSON" == "" ]; then
118         "${CURL[@]}" | $PYTHON -mjson.tool
119     else
120         "${CURL[@]}" << JSON_EOF | $PYTHON -mjson.tool
121     $JSON
122     JSON_EOF
123     fi
124 elif [ "$JSON" == "" ]; then
125     "${CURL[@]}"
126 else
127     "${CURL[@]}" << JSON_EOF
128     $JSON
129     JSON_EOF
130 fi
131 fi
132
133 echo ""
134

```

The following command line example shows how to retrieve detailed information for a specific user account using the `akrest` script.

```

$ ./akrest -u root -p verysecret 192.168.0.230 get user/v1/users/Edinburgh
{"user":
{"href": "/api/user/v1/users/Edinburgh",
"logname": "Edinburgh",
"fullname": "John Edinburgh",
"initial_password": "DummyPassword",
"require_annotation": false,
"roles": ["basic"],
"kiosk_mode": false,
"kiosk_screen": "status/dashboard",
"exceptions": [],
"preferences": {"href": "/api/user/v1/users/Edinburgh/preferences",
"locale": "C",
"login_screen": "status/dashboard",
"session_timeout": 15,
"advanced_analytics": false,
"keys": []
}
}}

```

Using Python

The Python code examples in this document heavily use the Python module structure. This enables creation of a library of commonly used functions for client code to access the RESTful API service in the Oracle ZFS Storage Appliance. Functions in Python RESTful API modules `restclientlib.py` and `restmulti.py` are made available to client code by importing the modules in client code modules using the Python `import` statement.

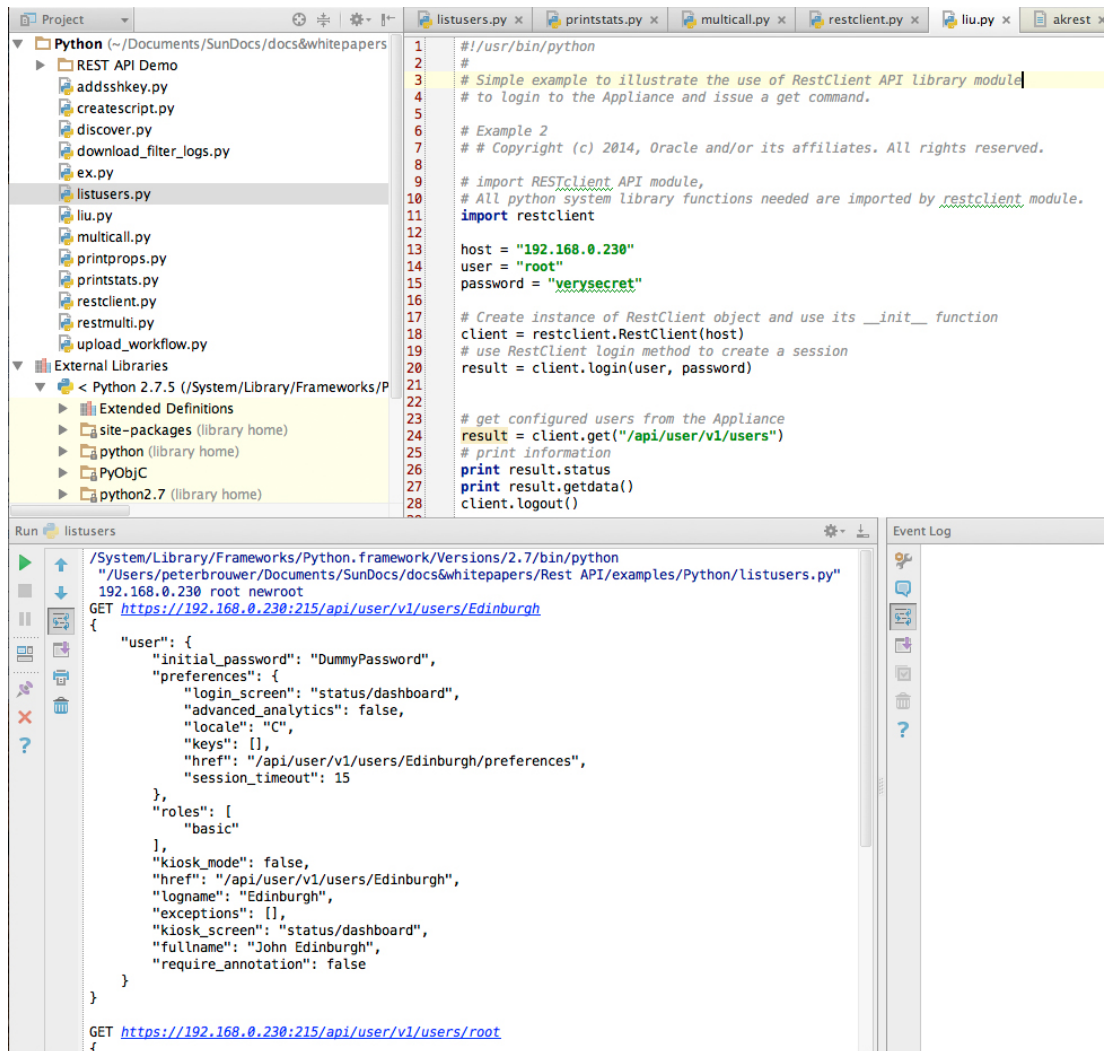
The code for the used Python Restful library modules `restclientlib` and `restmulti` in the following examples can be found in the appendices at the end of this document.

Python programming best practices

When writing Python code, try to write self-contained code modules, and avoid using global data variables. As Python is an Object Oriented type programming language, define data classes and implement methods (functions) operating on that data. The Python RESTful API modules can be used as examples.

Python code examples

The next example shows Python code, illustrating how to log in to the Oracle ZFS Storage Appliance and issue a GET command to retrieve its user accounts. Note that user and password login information is hard coded, which is not recommended in actual practice. A later section of this paper shows how to avoid including user names and passwords in code. The following illustration shows the code and part of its output.



```

1  #!/usr/bin/python
2  #
3  # Simple example to illustrate the use of RestClient API library module
4  # to login to the Appliance and issue a get command.
5
6  # Example 2
7  # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
8
9  # import RestClient API module,
10 # All python system library functions needed are imported by restclient module.
11 import restclient
12
13 host = "192.168.0.230"
14 user = "root"
15 password = "verysecret"
16
17 # Create instance of RestClient object and use its __init__ function
18 client = restclient.RestClient(host)
19 # use RestClient login method to create a session
20 result = client.login(user, password)
21
22
23 # get configured users from the Appliance
24 result = client.get("/api/user/v1/users")
25 # print information
26 print result.status
27 print result.getdata()
28 client.logout()

```

```

Run listusers
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python
"/Users/peterbrouwer/Documents/SunDocs/docs&whitepapers/Rest API/examples/Python/listusers.py"
192.168.0.230 root newroot
GET https://192.168.0.230:215/api/user/v1/users/Edinburgh
{
  "user": {
    "initial_password": "DummyPassword",
    "preferences": {
      "login_screen": "status/dashboard",
      "advanced_analytics": false,
      "locale": "C",
      "keys": [],
      "href": "/api/user/v1/users/Edinburgh/preferences",
      "session_timeout": 15
    },
    "roles": [
      "basic"
    ],
    "kiosk_mode": false,
    "href": "/api/user/v1/users/Edinburgh",
    "logname": "Edinburgh",
    "exceptions": [],
    "kiosk_screen": "status/dashboard",
    "fullname": "John Edinburgh",
    "require_annotation": false
  }
}
GET https://192.168.0.230:215/api/user/v1/users/root
{

```

Figure 5. Python code to log in and issue a get command for the Oracle ZFS Storage Appliance

The next step is to make the code in the example more generic and follow the Python module structure coding practice. A proper main function is defined, and if the module is started as a main module, the main function is called (code lines 44-46). Another change is the use of the `create_client` method of the `restclient` object. This method adds checks on arguments passed to it (code line 20).

An addition is the use of multithread functionality from the RESTful client API `restmulti` Python module. See the `restmulti` module import in line 12.

```

1  #!/usr/bin/python
2
3  # Example 3
4  # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
5  #
6  """An example of using multi-threaded requests to list the details for all
7  users in a system"""
8
9  import os
10 import sys
11 import restclientlib
12 import restmulti
13
14
15 def usage():
16     print "usage: python listusers.py <host> [user] [password]"
17     sys.exit(2)
18
19 def main(args):
20     client = restclientlib.create_client(*args)
21     if not client:
22         usage()
23
24     result = client.get("/api/user/v1/users")
25     if result.status == restclientlib.Status.OK:
26         users = result.getdata().get("users")
27         user_details = restmulti.RestMultiRequest()
28
29         # Create a multi-threaded request to get detail info for every user
30         for user in users:
31             request = restclientlib.RestRequest(
32                 "GET", "/api/user/v1/users/%s" % user.get("logname"))
33             user_details.add_request(client, request)
34
35         user_details.run()
36         user_details.wait()
37
38         # Print the results for listing all user details
39         for run in user_details.runs:
40             print str(run)
41
42     client.logout()
43
44 if __name__ == "__main__":
45     main(sys.argv[1:])
46     os._exit(0)

```

The next example demonstrates how to upload a workflow and use the option to pass on arguments to the workflow. Workflows are scripting code uploaded in the Oracle ZFS Storage Appliance and run under control of the Oracle ZFS Storage Appliance software shell. For more detailed information on workflows, see the white paper "Effectively Managing the Oracle ZFS Storage Appliance with Scripting" in the Oracle ZFS Storage Appliance White Papers web site listed in the References section.

The example shows an upload of a simple workflow that will stop after the number of seconds specified in the argument of the workflow. The Python script takes the workflow file name and a workflow parameter block passed as a JSON object.

The following is the workflow code:

```

1  # Example 4a
2  # Copyright (c) 2013, 2014 Oracle and/or its affiliates. All rights reserved.
3  # Workflow: slow_workflow.akwf
4
5  var workflow = {
6      name:          'Slow Return',
7      description:   'A workflow that takes a long time to end.',
8      scheduled:    false,
9      parameters:   {
10         seconds:   {
11             label: 'Seconds to sleep',
12             type:  'Integer'
13         },
14         sendOutput: {

```

```

15             label: 'Send output while executing',
16             type: 'Boolean'
17         }
18     },
19     execute: function (params) {
20         "use strict";
21         var i = 0;
22         for (i = 0; i < params.seconds; i = i + 1) {
23             run('sleep 1');
24             if (params.sendOutput) {
25                 printf('%s second\n', i);
26             }
27         }
28         return ('Workflow ended successfully.');
```

The workflow definition specifies the workflow characteristics. Note in code line 8 that `scheduled` is set to `false`, so the workflow can be executed using the RESTful API workflow `execute` function.

The Python module `upload_workflow` is used to upload the workflow (code line 89), pass on the parameters, and execute the workflow (code lines 101-112).

Note also the slightly different `import` syntax for the `restclientlib` module. With the python code from `<libmodulename> import *`, the classes and objects from that imported module can be referenced directly in the code. When using the `import <libmodulename>` syntax, a class from that module must be referred to as `<modulename>.<classname>`. Which method to use is a personal preference. When using multiple library modules, using the `<modulename>.` style of code writing makes it easier to track the location of classes and functions.

```

1  #!/usr/bin/python
2
3  # Example 4b
4  # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
5  #
6  """
7  Upload any workflow in your local folder/directory and run it using this script
8  Ensure that the workflow property "scheduled" is not set to true to execute
9  the workflow
10 """
11
12 from restclientlib import *
13 import getopt
14 import getpass
15 import sys
16 import json
17
18
19 def readfile(filename):
20     if "akwf" in filename.lower():
21         try:
22             with open(filename, "r") as f:
23                 return f.read()
24         except IOError as e:
25             print e
26     else:
27         print "Please upload an akwf file"
28
29
30 def usage():
31     print "upload_workflow.py      - Upload and Execute a workflow"
32     print "uses restclientlib.py - please ensure that it is in your workspace"
33     print "usage: upload_workflow.py [options] <zfsa-host>"
34     print "options:"
35     print "  -u <user> Login user. (default is root)"
36     print "  -p <pass> Login password."
37     print "  -f <filename> filename (neccessary)."
38     print "  -e <TRUE/FALSE> (default is false)."
39     print "  -c <JSON> (content to execute the workflow with). (optional)"
40
41
42 def main(argv):
43     do_execute = "False"
44     execute_content = ""
45     user = "root"
46     password = ""
47     filename = ""
48
49     try:
50         opts, args = getopt.getopt(argv[1:], "u:p:f:e:c:")
```

```

51     except getopt.GetoptError as err:
52         print str(err)
53         usage()
54         sys.exit(2)
55
56     for opt, arg in opts:
57         if opt == "-u":
58             user = arg
59         elif opt == "-p":
60             password = arg
61         elif opt == "-f":
62             filename = arg
63         elif opt == "-e":
64             do_execute = arg
65         elif opt == "-c":
66             execute_content = arg
67
68     if len(args) != 1:
69         print "Insufficient arguments"
70         usage()
71         sys.exit(2)
72
73     if not password:
74         password = getpass.getpass()
75
76     host = args[0]
77     client = RestClient(host)
78     result = client.login(user, password)
79
80     if result.status != Status.CREATED:
81         print "Login failed:"
82         print json.dumps(result.getdata(), sort_keys=True, indent=4)
83         sys.exit(1)
84
85     if filename == "":
86         print "Include a filename"
87
88     body = readfile(filename)
89     result = client.post("/api/workflow/v1/workflows", body)
90
91     if result.status != Status.CREATED:
92         print result.status
93         print result
94         raise Exception("Failed to upload the workflow")
95     else:
96         print "Workflow uploaded"
97         workflow = result.getdata()
98         print json.dumps(workflow, sort_keys=True, indent=4)
99         if do_execute.lower() == "true":
100             print execute_content
101             result = client.put(workflow["workflow"]["href"] + "/execute",
102                               execute_content)
103             if result.status != Status.ACCEPTED:
104                 print "The workflow cannot be executed. " \
105                       "Ensure that scheduled property is not set to true"
106                 print json.dumps(result.getdata(), sort_keys=True, indent=4)
107
108             else:
109                 print "The workflow has been executed"
110                 print "output:"
111                 print json.dumps(result.getdata(), sort_keys=True, indent=4)
112
113
114 if __name__ == "__main__":
115     main(sys.argv)
116

```

When executing the code, special attention needs to be given to the double quotes in the JSON formatted text block to pass the workflow parameters. Backslashes must be used to surround the double quotes required within the JSON text block so that the quotes are not stripped out by either the shell or IDE environment. The following figure shows how to do this using the PyCharm IDE.

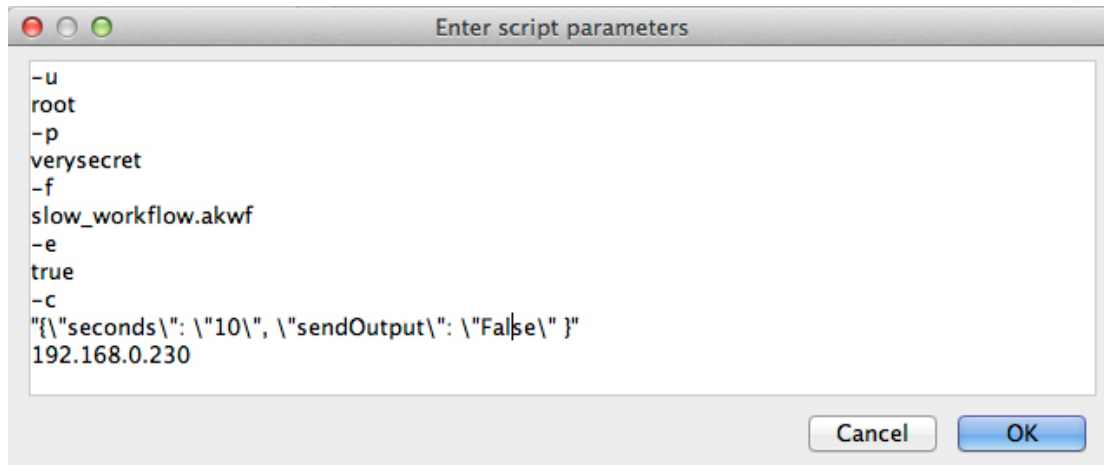


Figure 6. Using backslashes to prevent Python from stripping quotation marks in code when passed as an argument

Running the `upload_workflow` script generates the following output:

```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python
/Users/peterbrouwer/Documents/SunDocs/docs&whitepapers/Rest
API/examples/Python/upload_workflow.py -u root -p verysecret -f slow_workflow.akwf -e
true -c "{\"seconds\": \"10\", \"sendOutput\": \"False\"}" 192.168.0.230

Workflow uploaded
{
  "workflow": {
    "alert": false,
    "description": "A workflow that takes a long time to end.",
    "href": "/api/workflow/v1/workflows/5d29f146-0f52-6566-b443-f54eb11b5ea4",
    "name": "Slow Return",
    "origin": "<local>",
    "owner": "root",
    "scheduled": false,
    "setid": false,
    "uuid": "5d29f146-0f52-6566-b443-f54eb11b5ea4",
    "version": ""
  }
}
{"seconds": "10", "sendOutput": "False"}
The workflow has been executed
output:
{
  "result": "Workflow ended successfully.\n"
}

Process finished with exit code 0
```

The next example shows how to retrieve log information from the Oracle ZFS Storage Appliance. The Oracle ZFS Storage Appliance maintains status information classified according to severity (Alerts and Faults) and type (System and Audit). The Python module `download_filter_logs.py` uses the `-t` option (code line 50) to specify the type logs to be retrieved. Use the `-f` option to specify the name of the file in which to store the retrieved log info.

```
1  #!/usr/bin/python
2
3  # Example 5
4  #
5  # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
6  #
7
8
9  import restclientlib
10 import getopt
11 import getpass
12 import json
13 import sys
14
15
```

```

16 def usage():
17     print "download_filter_logs.py - Download and filter logs"
18     print "uses restclient.py - please ensure that it is in your workspace"
19     print "usage: download_logs [options] <zfsa-host>"
20     print "options:"
21     print "    -u <user> Login user. (default is root)"
22     print "    -p <pass> Login password."
23     print "    -t <logs type> (default is audit)"
24     print "    -f <filename> filename (default is logs.txt)."
25     print "    -F <filter> if -F is given. Login, Logouts entries will be" \
26         " deleted."
27     print "    only works if log type is audit"
28
29
30 def main(argv):
31     do_filter = False
32     filename = "logs.txt"
33     logtype = "audit"
34
35     user = "root"
36     password = ""
37
38     try:
39         opts, args = getopt.getopt(argv[1:], "u:p:t:f:F")
40     except getopt.GetoptError as err:
41         print str(err)
42         usage()
43         sys.exit(2)
44
45     for opt, arg in opts:
46         if opt == "-u":
47             user = arg
48         elif opt == "-p":
49             password = arg
50         elif opt == "-t":
51             logtype = arg
52         elif opt == "-f":
53             filename = arg
54         elif opt == "-F" and logtype == "audit":
55             do_filter = True
56
57     if len(args) != 1:
58         print "Insufficient arguments"
59         usage()
60         sys.exit(2)
61
62     if not password:
63         password = getpass.getpass()
64
65     host = args[0]
66     client = restclientlib.RestClient(host)
67
68     result = client.login(user, password)
69
70     if result.status != restclientlib.Status.CREATED:
71         print "Login failed:"
72         print json.dumps(result.getdata(), sort_keys=True, indent=4)
73         sys.exit(1)
74
75     download_log(client, logtype, filename)
76     if do_filter:
77         remove_login_logout(filename)
78
79
80 def download_log(client, logtype, filename):
81     result = client.get("/api/log/v1/collect/%s" % logtype)
82     if result.status != restclientlib.Status.OK:
83         raise Exception("failed to download the logs")
84     else:
85         fp = open('./%s' % filename, 'w')
86         line = result.readline()
87         while line:
88             fp.write(line)
89             line = result.readline()
90         fp.close()
91
92
93 def remove_login_logout(filename):
94     fp = open('./%s' % filename, 'r')
95     fpl = open('./%s.filtered' % filename, 'w')
96     lines = fp.readlines()
97     i = 0
98     while i < len(lines) - 1:
99         if "summary" in lines[i]:
100             if "User logged in" in lines[i] or "User logged out" in lines[i]:
101                 pass

```

```

102         else:
103             for j in range(-12, 2):
104                 fpl.write(lines[i+j])
105             i += 12
106         else:
107             i += 1
108     fp.close()
109     fpl.close()
110
111
112 if __name__ == "__main__":
113     main(sys.argv)
114

```

The last example demonstrates uploading an ssh key to the Oracle ZFS Storage Appliance to avoid having to code passwords into ssh-based scripts. The Python module `addsshkey.py` uses the file `authorized_keys` in the user's directory `~/ .ssh` (code line 73) to upload the ssh keys into the specified user's (code line 64) account of the Oracle ZFS Storage Appliance. The default used for user is `root` (code line 61).

First you need to create an SSH DSA-type key pair for authentication:

```

Peter-Brouwer-Mac-Pro: peterbrouwer$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/Users/peterbrouwer/.ssh/id_dsa):
/Users/peterbrouwer/.ssh/id_dsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/peterbrouwer/.ssh/id_dsa.
Your public key has been saved in /Users/peterbrouwer/.ssh/id_dsa.pub.
The key fingerprint is:
a5:68:a6:3b:7d:d5:12:1f:ef:40:8e:74:02:0c:f6:27 peterbrouwer@Peter-Brouwer-Mac-
Pro.local
The key's randomart image is:
+--[ DSA 1024]-----+
|          oo          |
|       .  .o          |
|      E.o            |
|     . =+ +         |
|    + A. Q o        |
|   +   # = .        |
|  ..   . . o        |
| ... .              |
| ..                |
+-----+

```

```
Peter-Brouwer-Mac-Pro: ~ peterbrouwer$
```

The Python `addsshkey` uses the file `authorized_keys` in the user's `~/ .ssh` directory to upload the keys, so add the just-generated key to that file:

```

Peter-Brouwer-Mac-Pro:~ peterbrouwer$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
Peter-Brouwer-Mac-Pro:~ peterbrouwer$

```

Now execute the Python `addsshkey.py` to upload the previously generated ssh key. After the upload, you can test the uploaded keys by using ssh to log in to the Oracle ZFS Storage Appliance. There should be no password request.

```

1  #!/usr/bin/python
2
3  # Example 6
4  # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
5  #
6
7  """Adds all public keys of the current user to an appliance"""
8
9  import getpass
10 import os
11 import restclientlib
12 import sys
13
14
15 def add_keys(appliance, user, password, filename):
16     """

```

```

17     Adds a ssh key to the specified appliance.
18
19     :param appliance: Host name
20     :param user: Appliance management login user name
21     :param password: User password
22     :param filename: Key filename
23     """
24
25     with open(filename) as key_file:
26         keys = key_file.readlines()
27     client = restclientlib.RestClient(appliance, user, password)
28
29     key_types = {
30         "ssh-dss": "DSA"
31     }
32
33     for k in keys:
34         words = k.split()
35         if len(words) != 3:
36             continue
37         key_type = key_types.get(words[0])
38         if not key_type:
39             continue
40         key = {
41             "type": key_type,
42             "key": words[1],
43             "comment": words[2]
44         }
45         path = "/api/user/v1/users/%s/preferences/keys" % user
46         result = client.post(path, key)
47         if result.status == 201:
48             print "Created key %s" % key
49         else:
50             print "Error creating %s\nError:%s" % (key, str(result))
51
52
53     def usage():
54         print "addsshkey.py - Add public SSH keys to an appliance user"
55         print "usage: python addsshkey.py <host> [user] [password]"
56         print "       If user is not supplied than 'root' is used as default"
57         print "       If password is not supplied then a prompt will be used"
58
59
60     def main():
61         user = "root"
62
63         if len(sys.argv) == 3:
64             user = sys.argv[2]
65         elif len(sys.argv) == 2:
66             pass
67         else:
68             print "usage: add_key.py <host> [user]"
69             sys.exit(2)
70
71         password = getpass.getpass()
72
73         filename = "%s/.ssh/authorized_keys" % os.environ['HOME']
74
75         print filename
76
77         add_keys(sys.argv[1], user, password, filename)
78
79
80     if __name__ == "__main__":
81         main()
82

```

Conclusion

The provided code examples in this paper have been written to illustrate the use of the RESTful API and in many cases lack full error checking on input parameters as well as detailed information on possible failing commands. Please use the examples accordingly. When creating programs in production environments, pay proper attention to writing code that fully checks user input and provides enough detail in diagnostic error messages for the user to understand the nature of a failure. A message such as 'Error encountered, contact your administrator' would not meet any standards of usefulness.

The RESTful API provides a full framework for administrators to create programs and scripts, tailored to the best practices and administrative procedures used within the organization, for addressing the Oracle ZFS Storage Appliance.

References

Oracle RESTful API documentation.

http://docs.oracle.com/cd/E51475_01/html/E52433/index.html

Oracle ZFS Storage Appliance Product Information

<http://www.oracle.com/us/products/servers-storage/storage/nas/overview/index.html>

Oracle ZFS Storage Appliance White Papers and Subject-Specific Resources

<http://www.oracle.com/technetwork/server-storage/sun-unified-storage/documentation/index.html>

Oracle ZFS Storage Appliance Document library

http://docs.oracle.com/cd/E51475_01/index.html

The *Oracle ZFS Storage Appliance Administration Guide* is also available through the Oracle ZFS Storage Appliance help context.

The Help function in Oracle ZFS Storage Appliance can be accessed through the browser user interface.

Python IDE environments

<https://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

Python

<https://www.python.org>

Appendix A: Python Code for `restmulty.py` Module

```

1  #!/usr/bin/python
2
3  # The sample code provided here is for training purposes only to help you to get
4  # familiar with the Oracle ZFS Storage Appliance RESTful API.
5  # As such the use of this code is unsupported and is for non-commercial or
6  # non-production use only.
7  # No effort has been made to include exception handling and error checking
8  # functionality as is required in a production environment.
9  #
10 # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
11 #
12
13 """Run many REST API client commands in parallel"""
14
15 import getopt
16 import json
17 import os
18 import restclientlib
19 import sys
20 import threading
21 import Queue
22
23 class _RestWorker(threading.Thread):
24     """A worker thread that runs REST API requests from a queue"""
25     def __init__(self, work_queue):
26         threading.Thread.__init__(self)
27         self._work_queue = work_queue # Queue containing requests
28         self._lock = threading.Lock() # Lock to protect properties below
29         self._request = None # Current REST request being processed
30         self._running = True # Worker will run while True
31         self.start() # Start this thread
32
33     def run(self):
34         """Run a REST API command from a queue. This method should only be
35         called by the thread that is running this worker via start()
36         """
37         with self._lock:
38             running = self._running
39
40         while running:
41             request = self._work_queue.get()
42             with self._lock:
43                 running = self._running
44                 if running:
45                     self._request = request
46
47             if running:
48                 try:
49                     self._request.run()
50                 except Exception as err:
51                     self._request.error = err
52
53             with self._lock:
54                 self._request = None
55                 running = self._running
56
57     def shutdown(self):
58         """Allows RestThreadPool to shutdown this thread."""
59         with self._lock:
60             self._running = False
61             if self._request:
62                 self._request.cancel()
63             self._request = None
64
65
66 class RestThreadPool(object):
67     """A pool of threads that will run REST API client requests."""
68     def __init__(self, max_threads=16):
69         """Creates a REST API thread pool.
70
71         :param max_threads: Max number of threads in the pool.
72         """
73         self._work_queue = Queue.Queue()
74         self._workers = list()
75         self.max_threads = max_threads
76
77     def add_request(self, *requests):
78         """Adds a REST API request to the thread pool queue to be processed"""
79         for request in requests:
80             self._work_queue.put(request)
81             num_threads = len(self._workers)

```

```

82         if self.max_threads <= 0 or self.max_threads > num_threads:
83             if self._work_queue.qsize() > num_threads:
84                 self._workers.append(_RestWorker(self._work_queue))
85
86     def stop(self):
87         """Stops all worker threads when thread pool is stopped"""
88         for worker in self._workers:
89             worker.shutdown()
90
91
92     class RestMultiRequest(object):
93         def __init__(self):
94             self.runs = list()
95
96         def add_request(self, client, request):
97             self.add_runner(restclientlib.RestRunner(client, request))
98
99         def add_runner(self, runner):
100            self.runs.append(runner)
101
102         def run(self, pool=None):
103             if not pool:
104                 pool = RestThreadPool()
105             pool.add_request(*self.runs)
106
107         def wait(self):
108             """Wait for all requests to finish"""
109             done = False
110             while not done:
111                 done = True
112                 for r in self.runs:
113                     if not r.result():
114                         done = False
115
116         def print_results(self):
117             """Print out all the response data from all of the requests"""
118             done = False
119             for r in self.runs:
120                 setattr(r, "print_results", False)
121             while not done:
122                 done = True
123                 for r in self.runs:
124                     if not r.print_results:
125                         if r.isdone():
126                             print r
127                             r.print_results = True
128                         else:
129                             done = False
130
131
132     #
133     # Main Program
134     #
135     def main(args):
136         verbose = False
137         pool = RestThreadPool()
138         default_user = "root"
139         default_password = ""
140         default_host = ""
141
142         try:
143             opts, args = getopt.getopt(args, "h:u:p:t:v")
144         except getopt.GetoptError as err:
145             print str(err)
146             usage()
147             sys.exit(2)
148
149         for opt, arg in opts:
150             if opt == "-t":
151                 pool.max_threads = int(arg)
152             elif opt == "-u":
153                 default_user = arg
154             elif opt == "-p":
155                 default_password = arg
156             elif opt == "-v":
157                 verbose = True
158             elif opt == "-h":
159                 default_host = arg
160
161         if len(args) != 1:
162             usage()
163             sys.exit(2)
164
165         data_file = args[0]
166
167         json_str = open(data_file).read()

```

```

168     json_data = json.loads(json_str)
169
170     request = RestMultiRequest()
171
172     def add_requests(config):
173         commands = config.get("commands")
174         if not commands:
175             return
176         host = config.get("host", default_host)
177         user = config.get("user", default_user)
178         password = config.get("password", default_password)
179         client = restclient.RestClient(host, user, password)
180         for command in commands:
181             req = restclient.RestRequest(*command)
182             runner = restclient.RestRunner(client, req, verbose=verbose)
183             request.add_runner(runner)
184
185     if isinstance(json_data, dict):
186         add_requests(json_data)
187     elif isinstance(json_data, list):
188         for c in json_data:
189             add_requests(c)
190
191     request.run(pool)
192     request.print_results()
193
194     failed = 0
195     succeeded = 0
196     tried = len(request.runs)
197     completed = 0
198
199     for r in request.runs:
200         result = r.result()
201         if result:
202             completed += 1
203             status = result.status
204             if status > 299 or status < 200:
205                 failed += 1
206             else:
207                 succeeded += 1
208
209     print "Completed %d of %d REST API calls" % (completed, tried)
210     print "Succeeded: %d" % succeeded
211     print "Failed: %d" % failed
212
213     os._exit(failed)
214
215
216     def usage():
217         print "restmulti.py - Make many REST API calls"
218         print "usage: restmulti.py [options] <config-file>"
219         print "options:"
220         print "    -t <threads> Max number of threads. (Default is 10)"
221         print "    -v          Turn on verbose output."
222         print "    -u <user>   Login user name"
223         print "    -p <passwd> Login user password"
224         print "    -h <host>   ZFSSA host"
225
226     if __name__ == "__main__":
227         try:
228             main(sys.argv[1:])
229         except KeyboardInterrupt:
230             os._exit(0)

```


Appendix B: Python Code for `restclient.py` Module

```

1  #!/usr/bin/python
2
3  # The sample code provided here is for training purposes only to help you to get
4  # familiar with the Oracle ZFS Storage Appliance RESTful API.
5  # As such the use of this code is unsupported and is for non-commercial or
6  # non-production use only.
7  # No effort has been made to include exception handling and error checking
8  # functionality as is required in a production environment.
9  #
10 # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
11 #
12
13 """A REST API client for the ZFSSA"""
14
15 import base64
16 import json
17 import httplib
18 import threading
19 import urllib2
20
21 class Status:
22     """Result HTTP Status"""
23
24     def __init__(self):
25         pass
26
27     OK = 200           #: Request return OK
28     CREATED = 201     #: New resource created successfully
29     ACCEPTED = 202    #: Command accepted
30     NO_CONTENT = 204 #: Command returned OK but no data will
31 be returned
32     BAD_REQUEST = 400           #: Bad Request
33     UNAUTHORIZED = 401         #: User is not authorized
34     FORBIDDEN = 403           #: The request is not allowed
35     NOT_FOUND = 404           #: The requested resource was not found
36     NOT_ALLOWED = 405         #: The request is not allowed
37     TIMEOUT = 408             #: Request timed out
38     CONFLICT = 409            #: Invalid request
39     BUSY = 503                #: Busy
40
41 class RestRequest(object):
42     def __init__(self, method, path, data=""):
43         self.method = method
44         self.data = data
45         if not path.startswith("/"):
46             path = "/" + path
47         if not path.startswith("/api"):
48             path = "/api" + path
49         self.path = path
50
51 class RestResult(object):
52     """Result from a REST API client operation"""
53
54     def __init__(self, response, error_status=0):
55         """Initialize a RestResult containing the results from a REST call"""
56         self.response = response
57         self.error_status = error_status
58         self._body = None
59
60     def __str__(self):
61         if self.error_status:
62             return str(self.response)
63
64         data = self.getdata()
65         if isinstance(data, (str, tuple)):
66             return data
67         return json.dumps(data, indent=4, default=str)
68
69     @property
70     def body(self):
71         """Get the entire returned text body. Will not return until all
72 data has been read from the server."""
73         self._body = ""
74         data = self.response.read()
75         while data:
76             self._body += data
77             data = self.response.read()
78         return self._body
79
80     @property

```

```

81     def status(self):
82         """Get the HTTP status result, or -1 if call failed"""
83         if self.error_status:
84             return self.error_status
85         else:
86             return self.response.getcode()
87
88     def readline(self):
89         """Reads a single line of data from the server. Useful for
90         commands that return streamed data.
91
92         :returns: A line of text read from the REST API server
93         """
94         if self.error_status:
95             return None
96         self.response.fp._rbufsize = 0
97         return self.response.readline()
98
99     def getdata(self):
100        """Get the returned data parsed into a python object. Right now
101        only supports JSON encoded data.
102
103        :return: Data is parsed as the returned data type into a python
104        object. If the data type isn't supported than the string value of
105        the data is returned.
106        """
107        if self.error_status:
108            return None
109        data = self.body
110        if data:
111            content_type = self.getheader("Content-Type")
112            if content_type.startswith("application/json"):
113                data = json.loads(data)
114        return data
115
116     def getheader(self, name):
117         """Get an HTTP header with the given name from the results
118
119         :param name: HTTP header name
120         :return: The header value or None if no value is found
121         """
122         if self.error_status:
123             return None
124         info = self.response.info()
125         return info.getheader(name)
126
127     def debug(self):
128         """Get debug text containing HTTP status and headers"""
129         if self.error_status:
130             return repr(self.response) + "\n"
131
132         msg = httplib.responses.get(self.status, "Unknown")
133         hdr = "HTTP/1.1 %d %s\n" % (self.status, msg)
134         return hdr + str(self.response.info())
135
136
137 class RestRunner(object):
138     """REST request runner for a background client call. Clients can obtain
139     the result when it is ready by calling result()
140     """
141     def __init__(self, client, request, **kwargs):
142         self._result = None # REST result from request
143         self._called = threading.Condition() # Result available condition
144         self.client = client # Client used to run request
145         self.request = request # REST Request
146         self.verbose = kwargs.get("verbose")
147
148     def __str__(self):
149         url = self.client.REST_URL % (self.client.host, self.request.path)
150         out = "%s %s %s\n" % (self.request.method, url, self.request.data)
151         if self.isdone():
152             if self.verbose:
153                 out += self._result.debug()
154                 out += "\n"
155                 out += str(self._result)
156                 out += "\n"
157             else:
158                 out += "waiting"
159         return out
160
161     def run(self):
162         """Thread run routine. Should only be called by thread"""
163         try:
164             result = self.client.execute(self.request)
165         except Exception as err:
166             result = RestResult(err, -1)

```

```

167         with self._called:
168             self._result = result
169             self._called.notify_all()
170
171     def isdone(self):
172         """Determine if the REST call has returned data.
173
174         :return: True if server has returned data, otherwise False
175         """
176         with self._called:
177             return self._result is not None
178
179     def result(self, timeout=0):
180         """Get the REST call result object once the call is finished.
181
182         :param timeout: The number of seconds to wait for the response to
183             finish
184         :returns: RestResult or None if not finished.
185         """
186         with self._called:
187             if self._result:
188                 return self._result
189             else:
190                 self._called.wait(timeout)
191                 return self._result
192
193     def cancel(self):
194         if self.isdone():
195             result = self.result()
196             if result:
197                 result.fp.close()
198
199
200 class RestClient(object):
201     """A REST Client API class to access the ZFSSA REST API"""
202     REST_URL = "https://%s:215%s"
203     ACCESS_URL = "https://%s:215/api/access/v1"
204
205     def __init__(self, host, user=None, password=None, session=None):
206         """Create a client that will communicate with the specified ZFSSA
207         host. If user and password are not supplied then the client must
208         login before making calls.
209
210         :param host: Appliance host name/ip address
211         :param user: Management user name
212         :param password: Management user password.
213         :param session: Create a client using an existing session
214         """
215         self.host = host
216         self.opener = urllib2.build_opener(urllib2.HTTPHandler)
217         self.services = None
218         if session:
219             self.opener.addheaders = [
220                 ("X-Auth-Session", session),
221                 ('Content-Type', 'application/json')]
222         elif user and password:
223             auth = "%s:%s" % (user, password)
224             basic = "Basic %s" % base64.encodestring(auth).replace('\n', '')
225             self.opener.addheaders = [
226                 ("Authorization", basic),
227                 ('Content-Type', 'application/json')]
228
229     def login(self, user, password):
230         """
231         Create a login session for a client. The client will keep track of
232         the login session information so additional calls can be made without
233         having to supply credentials.
234
235         :param user: The login user name
236         :param password: The ZFSSA user password
237         :return: The REST result of the login call
238         """
239         if self.services:
240             self.logout()
241
242         auth = "%s:%s" % (user, password)
243         basic = "Basic %s" % base64.encodestring(auth).replace('\n', '')
244         url = self.ACCESS_URL % self.host
245         request = urllib2.Request(url, '')
246         request.add_header('Authorization', basic)
247         request.get_method = lambda: 'POST'
248
249         try:
250             result = RestResult(self.opener.open(request))
251             if result.status == httplib.CREATED:
252                 session = result.getheader("X-Auth-Session")

```

```

253         self.opener.addheaders = [
254             ("X-Auth-Session", session),
255             ('Content-Type', 'application/json')]
256         data = result.getdata()
257         self.services = data["services"]
258     except urllib2.HTTPError as e:
259         result = RestResult(e)
260     return result
261
262 def logout(self):
263     """Logout of the appliance and clear session data"""
264     request = urllib2.Request(self.ACCESS_URL % self.host)
265     request.get_method = lambda: "DELETE"
266     result = self.call(request)
267     self.opener.addheaders = None
268     self.services = None
269     return result
270
271 def _service_url(self, module, version=None):
272     url = None
273     for service in self.services:
274         if module == service['name']:
275             if version and service['version'] != version:
276                 continue
277             url = service['uri']
278             break
279     return url
280
281 def url(self, path, **kwargs):
282     """
283     Get the URL of a resource path for the client.
284
285     :param path: Resource path
286     :key service: The name of the REST API service
287     :key version: The version of the service
288     :return:
289     """
290     service = kwargs.get("service")
291     if service:
292         url = self._service_url(service, kwargs.get("version")) + path
293     else:
294         url = self.REST_URL % (self.host, path)
295     return url
296
297 def call(self, request, background=False):
298     """Make a REST API call using the specified urllib2 request"""
299     if background:
300         runner = RestRunner(self, request)
301         thread = threading.Thread(target=runner)
302         thread.start()
303         return runner
304     try:
305         response = self.opener.open(request)
306         result = RestResult(response)
307     except urllib2.HTTPError as e:
308         result = RestResult(e)
309     return result
310
311 def get(self, path, **kwargs):
312     """Make a REST API GET call
313
314     :param path: Resource path
315     :return: RestResult
316     """
317     request = urllib2.Request(self.url(path, **kwargs))
318     return self.call(request, kwargs.get("background"))
319
320 def delete(self, path, **kwargs):
321     """Make a REST API DELETE call
322
323     :param path:
324     :return: RestResult
325     """
326     request = urllib2.Request(self.url(path, **kwargs))
327     request.get_method = lambda: "DELETE"
328     return self.call(request, kwargs.get("background"))
329
330 def put(self, path, data="", **kwargs):
331     """Make a REST API PUT call
332
333     :param path: Resource path
334     :param data: JSON input data
335     :return: RestResult
336     """
337     url = self.url(path, **kwargs)
338     if not isinstance(data, (str, unicode)):

```

```
339         data = json.dumps(data)
340         request = urllib2.Request(url, data)
341         request.get_method = lambda: "PUT"
342         request.add_header('Content-Type', "application/json")
343         return self.call(request, kwargs.get("background"))
344
345     def post(self, path, data="", **kwargs):
346         """Make a REST API POST call
347
348         :param path: Resource path
349         :param data: JSON input data
350         :return: RestResult
351         """
352         url = self.url(path, **kwargs)
353         if not isinstance(data, (str, unicode)):
354             data = json.dumps(data)
355         request = urllib2.Request(url, data)
356         request.get_method = lambda: "POST"
357         request.add_header('Content-Type', "application/json")
358         return self.call(request, kwargs.get("background"))
359     def execute(self, request, **kwargs):
360         """Make an HTTP REST request
361
362         :param method: HTTP command (GET, PUT, POST, DELETE)
363         :param path: Resource path
364         :param data: JSON input data
365         """
366         if request.method.lower() == "get":
367             return self.get(request.path, **kwargs)
368         if request.method.lower() == "put":
369             return self.put(request.path, request.data, **kwargs)
370         if request.method.lower() == "post":
371             return self.post(request.path, request.data, **kwargs)
372         if request.method.lower() == "delete":
373             return self.delete(request.path, **kwargs)
374         raise Exception(
375             "Invalid HTTP request '%s' "
376             "(Should be one of GET, PUT, POST, DELETE)" % request.method)
```



Working with the RESTful API for the Oracle ZFS
Storage Appliance

September 2014 Version 1.0

Authors: Peter Brouwer, Andrew Ness

Oracle Application Integration Engineering

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0611

Hardware and Software, Engineered to Work Together