Working with the RESTful API for the Oracle ZFS Storage Appliance

September, 2024, Version 2.0

Copyright © 2024, Oracle and/or its affiliates

Public

Purpose statement

This document provides guidance and best practices on how to integrate an Oracle ZFS Storage Appliance system into a network infrastructure, monitor its functioning, and troubleshoot any operational network problems.

Disclaimer

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

Copyright © 2024, Oracle and/or its affiliates

Table of contents

Copyright © 2024, Oracle and/or its affiliates

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.

Copyright © 2024, Oracle and/or its affiliates

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.

Copyright © 2024, Oracle and/or its affiliates

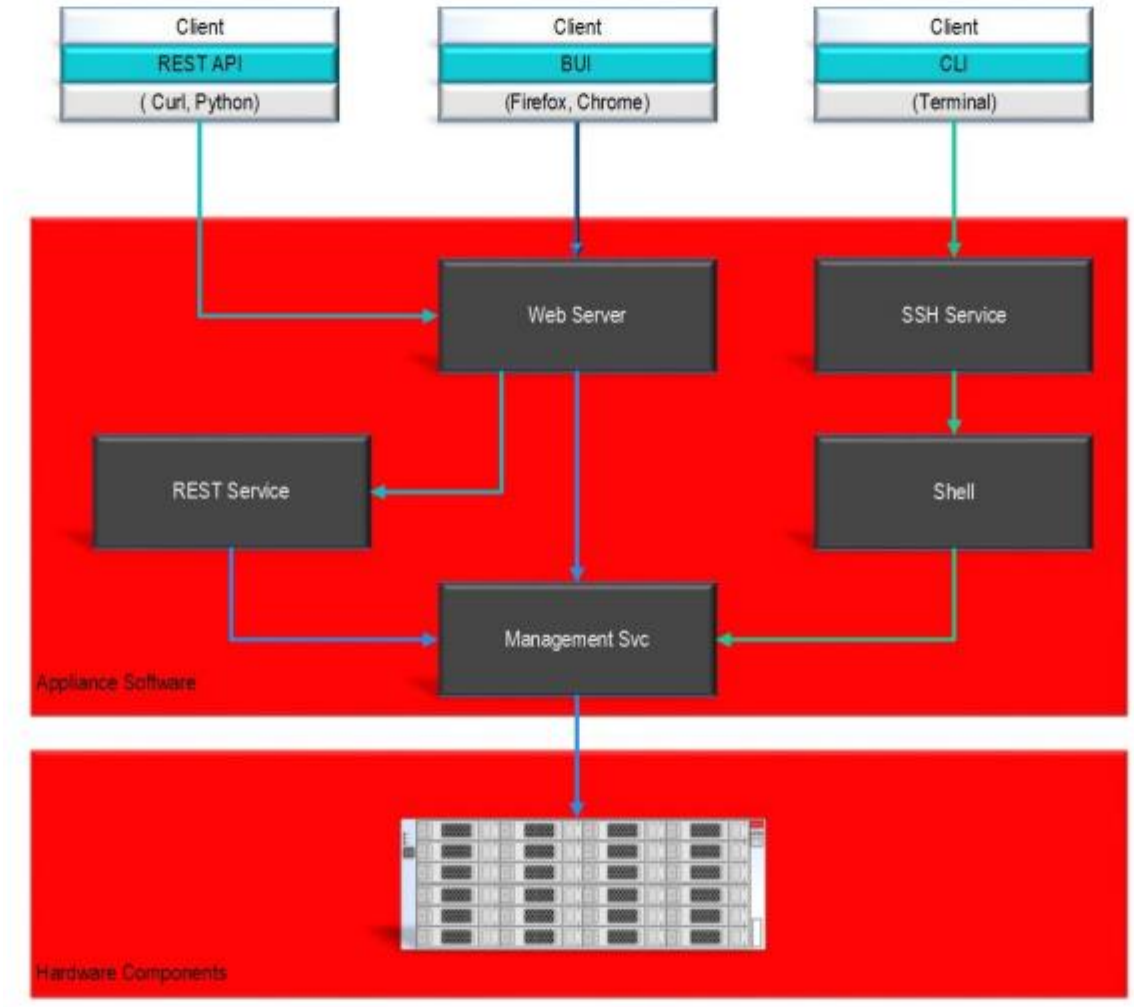


Figure 1. Client architecture for communicating with the Oracle ZFS Storage Appliance

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.

Copyright © 2024, Oracle and/or its affiliates

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.

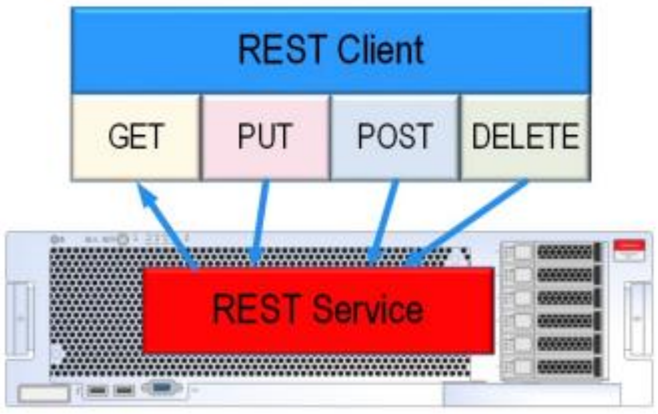


Figure 2. The REST Service operations

Table 1. CRUD Operations

Operation	Use
GET	List information about a resource – for example, storage pools, projects, LUNs, shares, and so on
POST	Create a new resource – `POST /storage/v1/pools` creates a new pool, for example
PUT	Modify a resource
DELETE	Destroy a resource

Copyright © 2024, Oracle and/or its affiliates

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

 }
```

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 CONTEST`	204	Command returned OK but no data will be returned

Copyright © 2024, Oracle and/or its affiliates

The following table defines some common error codes:

Table 3. Error Return Codes

Name	Code	Description
`ERR_INVALID_ARG`	400	Request returned success
`ERR_UNKNOWN_ARG`	400	New resource created successfully
`ERR_MISSING_ARG`	400	The request was accepted
`ERR_UNAUTHORIZED`	401	Command returned OK but no data will be returned
`ERR_DENIED`	403	The operation was 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
`ERR_NOT_IMPLEMENTED`	501	Operation not implemented
`ERR_BUSY`	503	Service not available due to limited resources

Copyright © 2024, Oracle and/or its affiliates

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)
```

Copyright © 2024, Oracle and/or its affiliates

```
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)
```

Copyright © 2024, Oracle and/or its affiliates

```
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
  ...
```

Copyright © 2024, Oracle and/or its affiliates

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.

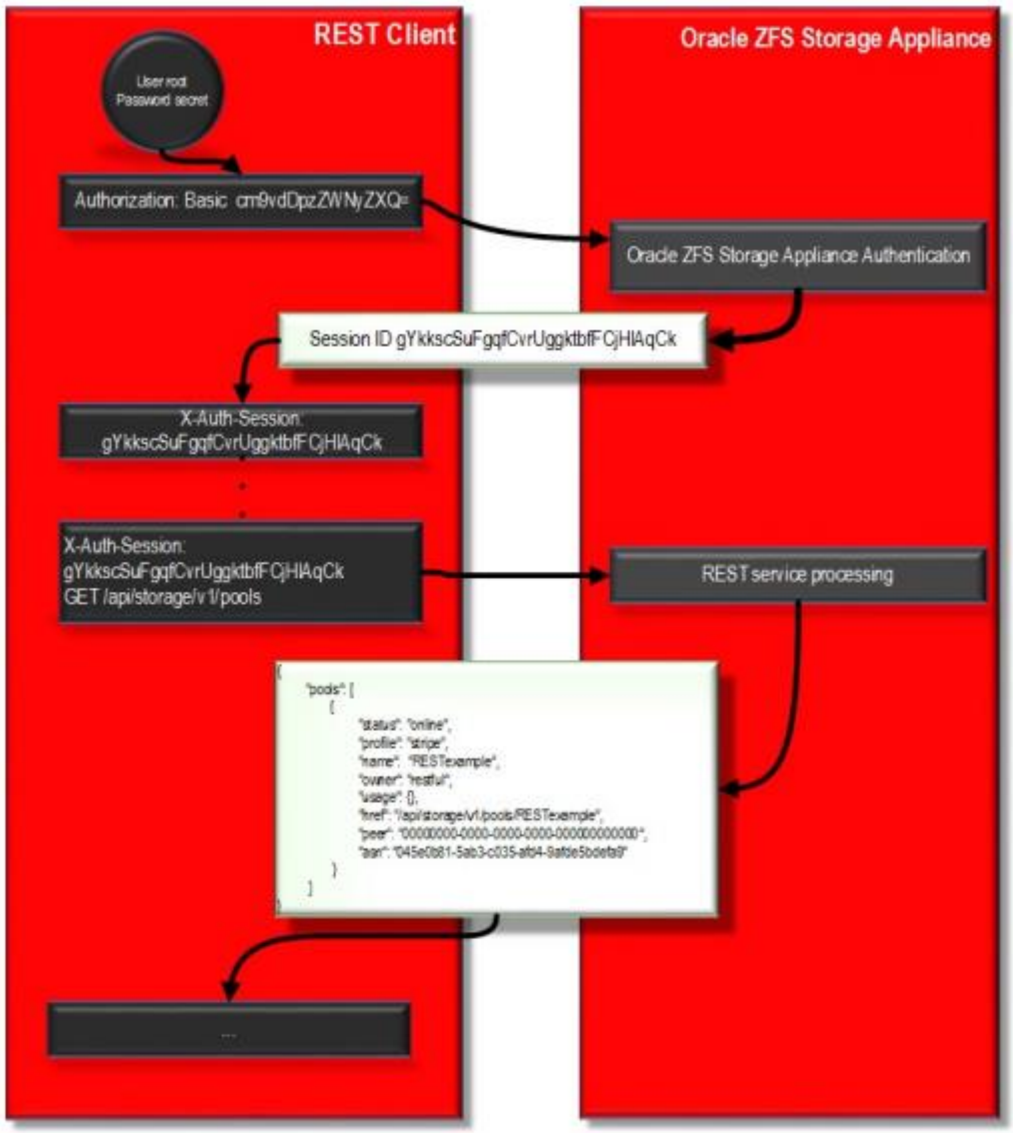
Authentication and Sessions

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.

Copyright © 2024, Oracle and/or its affiliates

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.


REST Client
Oracle ZFS Storage Appliance
User root
Password secret
Authorization: Basic cm9vdDpzZWNyZXQ=
Oracle ZFS Storage Appliance Authentication
Session ID gYkkscSuFgqfCvrUggktbfFCjHlAqCk
X-Auth-Session: gYkkscSuFgqfCvrUggktbfFCjHlAqCk
X-Auth-Session: gYkkscSuFgqfCvrUggktbfFCjHlAqCk
GET /api/storage/v1/pools
REST service processing
...

```
{
    "pools": [
        {
            "status": "online",
            "profile": "stripe",
            "name":  "RESTexample",
            "owner": "restful",
            "usage": {},
            "href": "/api/storage/v1/pools/RESTexample",
            "peer": "00000000-0000-0000-0000-000000000000",
            "asn": "045e0b81-5ab3-c035-afb4-9afde5bdefa9"
        }
    ]
}
```

Figure 3. Session variable use

Copyright © 2024, Oracle and/or its affiliates

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. Version 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

Copyright © 2024, Oracle and/or its affiliates

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.

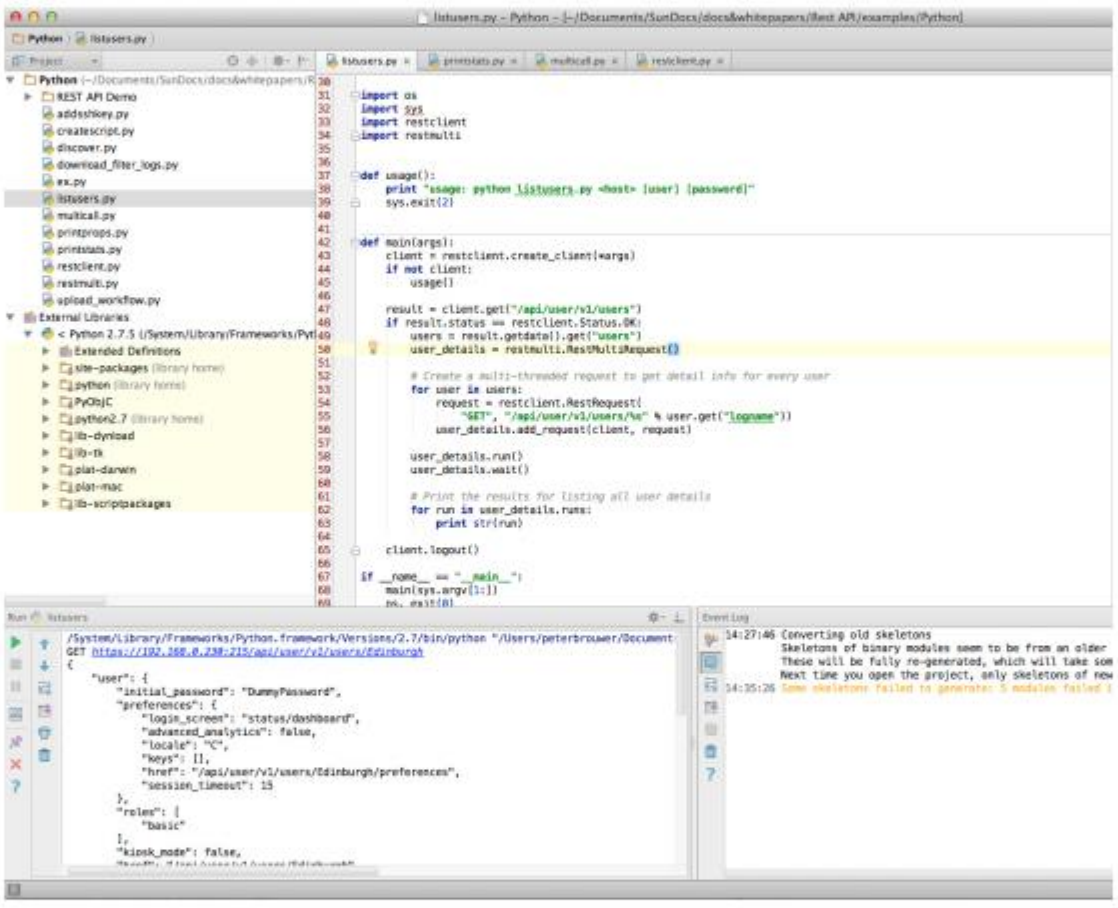
A key requirement is the support for JavaScript Object Notation (JSON) in the programming environment of choice. It is a lightweight data interchange format used by the RESTful API to exchange data between the client and the Oracle ZFS Storage Appliance.

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.

Copyright © 2024, Oracle and/or its affiliates



```
import os
import sys
import restclient
import restmulti


def usage():
    print "usage: python listusers.py <host> [user] [password]"
    sys.exit(2)


def main(args):
    client = restclient.create_client(*args)
    if not client:
        usage()

    result = client.get("/api/user/v1/users")
    if result.status == restclient.Status.OK:
        users = result.getdata().get("users")
        user_details = restmulti.RestMultiRequest()

        # Create a multi-threaded request to get detail info for every user
        for user in users:
            request = restclient.RestRequest(
                "GET", "/api/user/v1/users/%s" % user.get("logname"))
            user_details.add_request(client, request)

        user_details.run()
        user_details.wait()

        # Print the results for listing all user details
        for run in user_details.runs:
            print str(run)

    client.logout()

if __name__ == "__main__":
    main(sys.argv[1:])
    os._exit(0)
```

```
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python "/Users/peterbrouwer/Document
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,
```

```
14:27:46 Converting old skeletons
         Skeletons of binary modules seem to be from an older
         These will be fully re-generated, which will take som
         Next time you open the project, only skeletons of new
14:35:26 Some skeletons failed to generate: 5 modules failed t
```

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.

Copyright © 2024, Oracle and/or its affiliates

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*.

```
#!/bin/bash
#
# Example 1
# Copyright (c) 2013, 2014 Oracle and/or its affiliates. All rights reserved.
# Script akrest

CURL=(`which curl` -3 -k)        # curl command options
ACCEPT="application/json"        # Default returned content type accepted
DO_FORMAT=false                  # Pretty print JSON output
PYTHON=`which python`            # Used for pretty printing JSON output
USER=$ZFSSA_USER                 # Login user
PASSWORD=$ZFSSA_PASSWORD         # Login password
SESSION=$ZFSSA_SESSION           # Login session id
INFILE=                          # POST/PUT input file
CONTENT="application/json"       # Default input content type
VERBOSE=false                    # Print more data

usage() {
    echo "usage akrest [options] <host> <get|post|put|delete> <path> [json]"
    echo "options:"
    echo "    -f Format output"
    echo "    -h Print headers"
    echo "    -c Request CLI script"
    echo "    -i <file> Input file to post/put"
    echo "    -s <id>   Session id"
    echo "    -p <pass> Login password"
    echo "    -u <user> Login username"
    echo "    -v Verbose"
    echo "    -y Request YAML output"
    echo "    -z Request compressed return data (only some commands supported)"
    exit 2
}

while getopts u:p:i:s:hvbcfyz name
    do
    case $name in
    c)  CURL=( "${CURL[@]}" "--header" "X-Zfssa-Get-Script: true" );;
    b)  CONTENT="application/octet-stream";;
    f)  DO_FORMAT="true";;
    u)  USER="$OPTARG";;
    p)  PASSWORD="$OPTARG"
        SESSION=;;
    h)  CURL=( "${CURL[@]}" "-i" );;
    i)  INFILE=$OPTARG;;
    s)  CURL=( "${CURL[@]}" --header "X-Auth-Session: $OPTARG" )
        PASSWORD="";;
    v)  VERBOSE="true"
        CURL=( "${CURL[@]}" "-v" );;
    y)  ACCEPT="text/x-yaml";;
    z)  CURL=( "${CURL[@]}" "--header" "Accept-Encoding: gzip" );;
    ?)  usage
    esac
done
shift $(($OPTIND - 1))

if [ "$#" == "3" ]; then
    JSON=""
elif [ "$#" == "4" ]; then
    JSON=$4
    CURL=( "${CURL[@]}" "-d" "@-" "--header" "Content-Type: ${CONTENT}")
else
    usage
fi

HOST=$1
```

Copyright © 2024, Oracle and/or its affiliates

```
REQUEST=$2
PATH=$3
DATA=$4

case $REQUEST in
    get) REQUEST=GET;;
    put) REQUEST=PUT;;
    post) REQUEST=POST;;
    delete) REQUEST=DELETE;;
    *) usage
esac

if [ "$HOST" == "" ]; then
    usage
fi
if [ "$PATH" == "" ]; then
    usage
fi
if [ "localhost" == "$HOST" ]; then
    URL=http://$HOST:8215/$PATH
else
    URL=https://$HOST:215/api/$PATH
fi

if [ "${USER}" == "" ]; then
    USER=root
fi
if [ "${SESSION}" != "" ]; then
    CURL=("${CURL[@]}" --header "X-Auth-Session: ${SESSION}")
elif [ "${PASSWORD}" != "" ]; then
    CURL=("${CURL[@]}" --user "${USER}:${PASSWORD}")
else
    if [ "$HOST" != "localhost" ]; then
        echo "Either password or session needs to be set"
        exit 1
    fi
fi
```

Copyright © 2024, Oracle and/or its affiliates

```

if [ "${INFILE}" == "" ]; then
    CURL=( "${CURL[@]}" "-sS" )
else
    CURL=( "${CURL[@]}" "-d" "@${INFILE}" "--header" "Content-Type: $CONTENT" )
fi

CURL=("${CURL[@]}" "--header" "Accept: ${ACCEPT}" -X "${REQUEST}" "${URL}")

if [ "${VERBOSE}" == "true" ]; then
    echo "${CURL[@]}"
fi

if [ "${DO_FORMAT}" == "true" ]; then
    if [ "$JSON" == "" ]; then
        "${CURL[@]}" | $PYTHON -mjson.tool
    else
        "${CURL[@]}" << JSON_EOF | $PYTHON -mjson.tool
$JSON
JSON_EOF
    fi
elif [ "$JSON" == "" ]; then
    "${CURL[@]}"
else

    "${CURL[@]}" << JSON_EOF
$JSON
JSON_EOF
fi

echo ""

```

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

Copyright © 2024, Oracle and/or its affiliates

```
$ ./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.

Copyright © 2024, Oracle and/or its affiliates

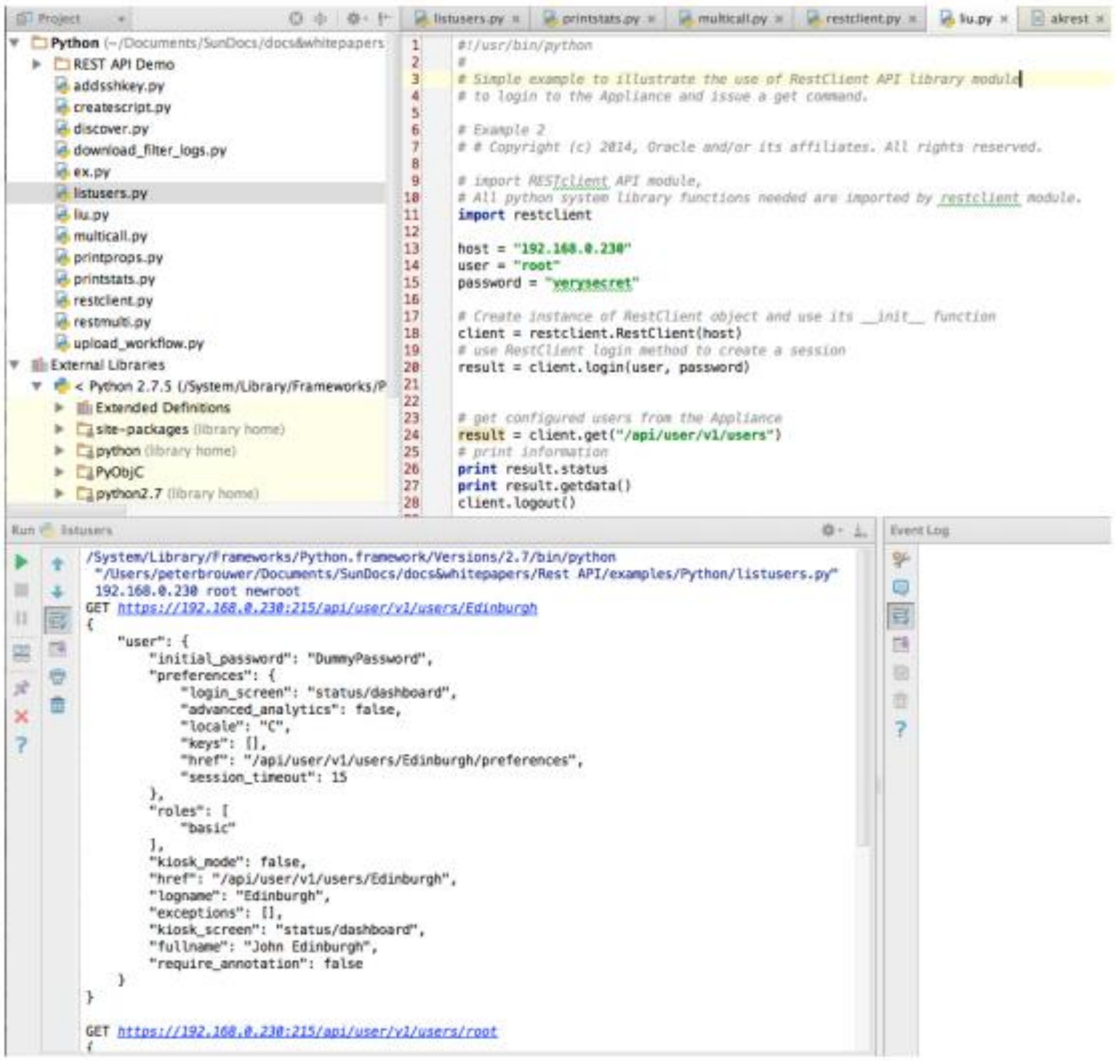
Python programming best practices

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.

Copyright © 2024, Oracle and/or its affiliates



```python
#!/usr/bin/python
#
# Simple example to illustrate the use of RestClient API library module
# to login to the Appliance and issue a get command.

# Example 2
# # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.

# import RESTclient API module,
# All python system library functions needed are imported by restclient module.
import restclient

host = "192.168.0.230"
user = "root"
password = "verysecret"

# Create instance of RestClient object and use its __init__ function
client = restclient.RestClient(host)
# use RestClient login method to create a session
result = client.login(user, password)


# get configured users from the Appliance
result = client.get("/api/user/v1/users")
# print information
print result.status
print result.getdata()
client.logout()
```

```
/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 technical paper "Effectively Managing the Oracle ZFS Storage Appliance with Scripting" in the Oracle ZFS Storage Appliance Technical Papers web site listed in the References section.

Copyright © 2024, Oracle and/or its affiliates

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:

```
# Example 4a
# Copyright (c) 2013, 2014 Oracle and/or its affiliates. All rights reserved.
# Workflow: slow_workflow.akwf

var workflow = {
    name:          'Slow Return',
    description:   'A workflow that takes a long time to end.',
    scheduled:     false,
    parameters: {
                    seconds: {
                        label: 'Seconds to sleep',
                        type: 'Integer'
                    },
                    sendOutput: {
                        label: 'Send output while executing',
                        type: 'Boolean'
        }
    },
    execute: function (params) {
        "use strict";
        var i = 0;
        for (i = 0; i < params.seconds; i = i + 1) {
            run('sleep 1');
            if (params.sendOutput) {
                printf('%s second\n', i);
            }
        }
        return ('Workflow ended successfully.');
    }
};

```

Copyright © 2024, Oracle and/or its affiliates

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 jason
17
18
```

```
def readfile(filename):
    if "akwf" in filename.lower():
        try:
            with open(filename, "r") as f:
                return f.read()
        except IOError as e:
            print e
    else:
        print "Please upload an akwf file"


def usage():
    print "upload_workflow.py     - Upload and Execute a workflow"
    print "uses restclientlib.py  - please ensure that it is in your workspace"
    print "usage: upload_workflow.py [options] <zfssa-host>"
    print "options:"
    print "    -u <user> Login user.  (default is root)"
    print "    -p <pass> Login password."
    print "    -f <filename> filename (neccessary)."
    print "    -e <TRUE/FALSE> (default is false)."
    print "    -c <JSON> (content to execute the workflow with). (optional)"


def main(argv):
    do_execute = "False"
    execute_content = ""
    user = "root"
    password = ""
    filename = ""

    try:
         opts, args = getopt.getopt(argv[1:], "u:p:f:e:c:")
    except getopt.GetoptError as err:
        print str(err)
        usage()
        sys.exit(2)
```

Copyright © 2024, Oracle and/or its affiliates

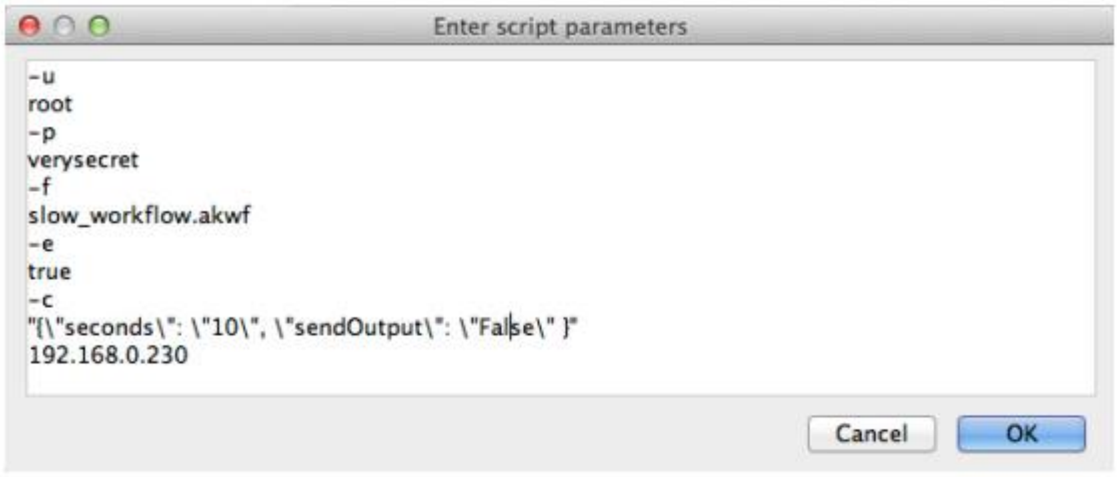
```

    for opt, arg in opts:
        if opt == "-u":
            user = arg
        elif opt == "-p":
            password = arg
        elif opt == "-f":
            filename = arg
        elif opt == "-e":
            do_execute = arg
        elif opt == "-c":
            execute_content = arg

    if len(args) != 1:
        print "Insufficient arguments"
        usage()
        sys.exit(2)

    if not password:
        password = getpass.getpass()

    host = args[0]
    client = RestClient(host)
    result = client.login(user, password)

    if result.status != Status.CREATED:
        print "Login failed:"
        print json.dumps(result.getdata(), sort_keys=True, indent=4)
        sys.exit(1)
```

Copyright © 2024, Oracle and/or its affiliates

```

    if filename == "":
        print "Include a filename"

    body = readfile(filename)
    result = client.post("/api/workflow/v1/workflows", body)

    if result.status != Status.CREATED:
        print result.status
        print result
        raise Exception("Failed to upload the workflow")
    else:
        print "Workflow uploaded"
        workflow = result.getdata()
        print json.dumps(workflow, sort_keys=True, indent=4)
        if do_execute.lower() == "true":
            print execute_content
            result = client.put(workflow["workflow"]["href"] + "/execute",
                                execute_content)
            if result.status != Status.ACCEPTED:
                print "The workflow cannot be executed. " \
                      "Ensure that scheduled property is not set to true"
                print json.dumps(result.getdata(), sort_keys=True, indent=4)

            else:
                print "The workflow has been executed"
                print "output:"
                print json.dumps(result.getdata(), sort_keys=True, indent=4)


if __name__ == "__main__":
    main(sys.argv)

```

Copyright © 2024, Oracle and/or its affiliates

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.



```
-u
root
-p
verysecret
-f
slow_workflow.akwf
-e
true
-c
"{\"seconds\": \"10\", \"sendOutput\": \"False\" }"
192.168.0.230
```

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:

Copyright © 2024, Oracle and/or its affiliates

```
/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.

Copyright © 2024, Oracle and/or its affiliates

```
   #!/usr/bin/python

   # Example 5
   #
   # Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
   #


   import restclientlib
   import getopt
   import getpass
   import json
   import sys


def usage():
       print "download_filter_logs.py - Download and filter logs"
       print "uses restclient.py - please ensure that it is in your workspace"
       print "usage: download_logs [options] <zfssa-host>"
       print "options:"
       print "    -u <user> Login user.  (default is root)"
       print "    -p <pass> Login password."
       print "    -t <logs type> (default is audit)"
       print "    -f <filename> filename (default is logs.txt)."
       print "    -F <filter> if -F is given. Login, Logouts entries will be" \
             " deleted."
       print "              only works if log type is audit"

```

Copyright © 2024, Oracle and/or its affiliates

```python

def main(argv):
    do_filter = False
    filename = "logs.txt"
    logtype = "audit"

    user = "root"
    password = ""

    try:
        opts, args = getopt.getopt(argv[1:], "u:p:t:f:F")
    except getopt.GetoptError as err:
        print str(err)
        usage()
        sys.exit(2)

    for opt, arg in opts:
        if opt == "-u":
            user = arg
        elif opt == "-p":
            password = arg
        elif opt == "-t":
            logtype = arg
        elif opt == "-f":
            filename = arg
        elif opt == "-F" and logtype == "audit":
            do_filter = True

    if len(args) != 1:
        print "Insufficient arguments"
        usage()
        sys.exit(2)

    if not password:
        password = getpass.getpass()

    host = args[0]
```

Copyright © 2024, Oracle and/or its affiliates

```
    client = restclientlib.RestClient(host)

    result = client.login(user, password)

    if result.status != restclientlib.Status.CREATED:
        print "Login failed:"
        print json.dumps(result.getdata(), sort_keys=True, indent=4)
        sys.exit(1)

    download_log(client, logtype, filename)
    if do_filter:
        remove_login_logout(filename)


def download_log(client, logtype, filename):
    result = client.get("/api/log/v1/collect/%s" % logtype)
    if result.status != restclientlib.Status.OK:
        raise Exception("failed to download the logs")
    else:
        fp = open('./%s' % filename, 'w')
        line = result.readline()
        while line:
            fp.write(line)
            line = result.readline()
        fp.close()


def remove_login_logout(filename):
    fp = open('./%s' % filename, 'r')
    fp1 = open('./%s.filtered' % filename, 'w')
    lines = fp.readlines()
    i = 0
    while i < len(lines) - 1:
        if "summary" in lines[i]:
            if "User logged in" in lines[i] or "User logged out" in lines[i]:
                pass
```

Copyright © 2024, Oracle and/or its affiliates

```
            else:
                for j in range(-12, 2):
                    fp1.write(lines[i+j])
            i += 12
        else:
            i += 1
    fp.close()
    fp1.close()


if __name__ == "__main__":
    main(sys.argv)

```

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).

Copyright © 2024, Oracle and/or its affiliates

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$
```

Copyright © 2024, Oracle and/or its affiliates

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` script 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.

```
#!/usr/bin/python

# Example 6
# Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
#

"""Adds all public keys of the current user to an appliance"""

import getpass
import os
import restclientlib
import sys


def add_keys(appliance, user, password, filename):

    Adds a ssh key to the specified appliance.

    :param appliance: Host name
    :param user: Appliance management login user name
```

Copyright © 2024, Oracle and/or its affiliates

```
    :param password: User password
    :param filename: Key filename
    """

    with open(filename) as key_file:
        keys = key_file.readlines()
    client = restclientlib.RestClient(appliance, user, password)

    key_types = {
        "ssh-dss": "DSA"
    }

    for k in keys:
        words = k.split()
        if len(words) != 3:
            continue
        key_type = key_types.get(words[0])
        if not key_type:
            continue
        key = {
            "type": key_type,
            "key": words[1],
            "comment": words[2]
        }
        path = "/api/user/v1/users/%s/preferences/keys" % user
        result = client.post(path, key)
        if result.status == 201:
            print "Created key %s" % key
        else:
            print "Error creating %s\nError:%s" % (key, str(result))

```

Copyright © 2024, Oracle and/or its affiliates

```python

def usage():
    print "addsshkey.py - Add public SSH keys to an appliance user"
    print "usage: python addsshkey.py <host> [user] [password]"
    print "       If user is not supplied than 'root' is used as default"
    print "       If password is not supplied then a prompt will be used"


def main():
    user = "root"

    if len(sys.argv) == 3:
        user = sys.argv[2]
    elif len(sys.argv) == 2:
        pass
    else:
        print "usage: add_key.py <host> [user]"
        sys.exit(2)

    password = getpass.getpass()

    filename = "%s/.ssh/authorized_keys" % os.environ['HOME']

    print filename

    add_keys(sys.argv[1], user, password, filename)


if __name__ == "__main__":
    main()

```

Copyright © 2024, Oracle and/or its affiliates

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
https://docs.oracle.com/en/storage/zfs-storage/zfs-appliance/os8-8-x/restful-api-guide/index.html

Oracle ZFS Storage Appliance Product Information
https://www.oracle.com/storage/nas/

Oracle ZFS Storage Appliance Technical Papers and Subject-Specific Resources
https://www.oracle.com/storage/technologies/nas-unified-storage-documentation.html

Oracle ZFS Storage Appliance Document library
https://docs.oracle.com/cd/F24627_01/

Python IDE environments
https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Python
https://www.python.org

Copyright © 2024, Oracle and/or its affiliates

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

```
#!/usr/bin/python

# The sample code provided here is for training purposes only to help you to get
# familiar with the Oracle ZFS Storage Appliance RESTful API.
# As such the use of this code is unsupported and is for non-commercial or
# non-production use only.
# No effort has been made to include exception handling and error checking
# functionality as is required in a production environment.
#
# Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
#

"""Run many REST API client commands in parallel"""

import getopt
import json
import os
import restclientlib
import sys
import threading
import Queue

class _RestWorker(threading.Thread):
    """A worker thread that runs REST API requests from a queue"""
    def _init_(self, work_queue):
        threading.Thread._init_(self)
        self._work_queue = work_queue   # Queue containing requests
        self._lock = threading.Lock()   # Lock to protect properties below
        self._request = None            # Current REST request being processed
        self._running = True            # Worker will run while True
        self.start()                    # Start this thread
```

```python

    def run(self):
        """Run a REST API command from a queue.  This method should only be
        called by the thread that is running this worker via start()
        """
        with self._lock:
            running = self._running

        while running:
            request = self._work_queue.get()
            with self._lock:
                running = self._running
                if running:
                    self._request = request

            if running:
                try:
                    self._request.run()
                except Exception as err:
                    self._request.error = err

            with self._lock:
                self._request = None
                running = self._running

    def shutdown(self):
        """Allows RestThreadPool to shutdown this thread."""
        with self._lock:
            self._running = False
            if self._request:
                self._request.cancel()
            self._request = None


class RestThreadPool(object):
    """A pool of threads that will run REST API client requests."""
    def _init_(self, max_threads=16):
```

Copyright © 2024, Oracle and/or its affiliates

```
        """Creates a REST API thread pool.

        :param max_threads: Max number of threads in the pool.
        """
        self._work_queue = Queue.Queue()
        self._workers = list()
        self.max_threads = max_threads

    def add_request(self, *requests):
        """Adds a REST API request to the thread pool queue to be
processed"""
        for request in requests:
            self._work_queue.put(request)
            num_threads = len(self._workers)
          if self.max_threads <= 0 or self.max_threads > num_threads:
                if self._work_queue.qsize() > num_threads:
                    self._workers.append(_RestWorker(self._work_queue))

    def stop(self):
        """Stops all worker threads when thread pool is stopped"""
        for worker in self._workers:
            worker.shutdown()


class RestMultiRequest(object):
    def __init__(self):
        self.runs = list()

    def add_request(self, client, request):
        self.add_runner(restclientlib.RestRunner(client, request))

    def add_runner(self, runner):
        self.runs.append(runner)

    def run(self, pool=None):
        if not pool:
            pool = RestThreadPool()
```

Copyright © 2024, Oracle and/or its affiliates

```
        pool.add_request(*self.runs)

    def wait(self):
        """Wait for all requests to finish"""
        done = False
        while not done:
            done = True
            for r in self.runs:\
                if not r.result():
                    done = False

    def print_results(self):
        """Print out all the response data from all of the requests"""
        done = False
        for r in self.runs:
            setattr(r, "print_results", False)
        while not done:
            done = True
            for r in self.runs:
                if not r.print_results:
                    if r.isdone():
                        print r
                        r.print_results = True
                    else:
                        done = False


#
# Main Program
#
def main(args):
    verbose = False
    pool = RestThreadPool()
    default_user = "root"
    default_password = ""
    default_host = ""

```

Copyright © 2024, Oracle and/or its affiliates

```
    try:
        opts, args = getopt.getopt(args, "h:u:p:t:v")
    except getopt.GetoptError as err:
        print str(err)
        usage()
        sys.exit(2)

    for opt, arg in opts:
        if opt == "-t":
            pool.max_threads = int(arg)
        elif opt == "-u":
            default_user = arg
        elif opt == "-p":
            default_password = arg
        elif opt == "-v":
            verbose = True
        elif opt == "-h":
            default_host = arg

    if len(args) != 1:
        usage()
        sys.exit(2)

    data_file = args[0]

json_str = open(data_file).read()
json_data = json.loads(json_str)

request = RestMultiRequest()

def add_requests(config):
    commands = config.get("commands")
    if not commands:
        return
    host = config.get("host", default_host)
    user = config.get("user", default_user)
    password = config.get("password", default_password)
```

Copyright © 2024, Oracle and/or its affiliates

```
    client = restclient.RestClient(host, user, password)
    for command in commands:
        req = restclient.RestRequest(*command)
        runner = restclient.RestRunner(client, req, verbose=verbose)
        request.add_runner(runner)

if isinstance(json_data, dict):
    add_requests(json_data)
elif isinstance(json_data, list):
    for c in json_data:
        add_requests(c)

request.run(pool)
request.print_results()

failed = 0
succeeded = 0
tried = len(request.runs)
completed = 0

for r in request.runs:
    result = r.result()
    if result:
        completed += 1
        status = result.status
        if status > 299 or status < 200:
            failed += 1
        else:
            succeeded += 1

print "Completed %d of %d REST API calls" % (completed, tried)
print "Succeeded: %d" % succeeded
print "Failed: %d" % failed

os._exit(failed)
```

Copyright © 2024, Oracle and/or its affiliates

```
def usage():
    print "restmulti.py - Make many REST API calls"
    print "usage: restmulti.py [options] <config-file>"
    print "options:"
    print "    -t <threads> Max number of threads. (Default is 10)"
    print "    -v          Turn on verbose output."
    print "    -u <user>    Login user name"
    print "    -p <passwd>  Login user password"
    print "    -h <host>    ZFSSA host"

if _name_ == "_main_":
    try:
        main(sys.argv[1:])
    except KeyboardInterrupt:
        os._exit(0)
```

Copyright © 2024, Oracle and/or its affiliates

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

```
#!/usr/bin/python

# The sample code provided here is for training purposes only to help you to get
# familiar with the Oracle ZFS Storage Appliance RESTful API.
# As such the use of this code is unsupported and is for non-commercial or
# non-production use only.
# No effort has been made to include exception handling and error checking
# functionality as is required in a production environment.
#
# Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
#

"""A REST API client for the ZFSSA"""

import base64
import json
import httplib
import threading
import urllib2

class Status:
    """Result HTTP Status"""

    def _init_(self):
        pass

    OK = 200                        #: Request return OK
    CREATED = 201                   #: New resource created successfully
    ACCEPTED = 202                  #: Command accepted
    NO_CONTENT = 204                #: Command returned OK but no data
will
be returned
    BAD_REQUEST = 400               #: Bad Request
    UNAUTHORIZED = 401              #: User is not authorized
    FORBIDDEN = 403                 #: The request is not allowed
```

Copyright © 2024, Oracle and/or its affiliates

```
    NOT_FOUND = 404                   #: The requested resource was not found
    NOT_ALLOWED = 405                 #: The request is not allowed
    TIMEOUT = 408                     #: Request timed out
    CONFLICT = 409                    #: Invalid request
    BUSY = 503                        #: Busy

class RestRequest(object):
    def _init_(self, method, path, data=""):
        self.method = method
        self.data = data
        if not path.startswith("/"):
            path = "/" + path
        if not path.startswith("/api"):
            path = "/api" + path
        self.path = path


class RestResult(object):
    """Result from a REST API client operation"""

    def _init_(self, response, error_status=0):
        """Initialize a RestResult containing the results from a REST call"""
        self.response = response
        self.error_status = error_status
        self._body = None

    def _str_(self):
        if self.error_status:
            return str(self.response)

        data = self.getdata()
        if isinstance(data, (str, tuple)):
            return data
        return json.dumps(data, indent=4, default=str)

    @property
    def body(self):
```

Copyright © 2024, Oracle and/or its affiliates

```
        """Get the entire returned text body.  Will not return until all
        data has been read from the server."""
        self._body = ""
        data = self.response.read()
        while data:
            self._body += data
            data = self.response.read()
        return self._body

    @property
    def status(self):
        """Get the HTTP status result, or -1 if call failed"""
        if self.error_status:
            return self.error_status
        else:
            return self.response.getcode()

    def readline(self):
        """Reads a single line of data from the server.  Useful for
        commands that return streamed data.

        :returns: A line of text read from the REST API server
        """
        if self.error_status:
            return None
        self.response.fp._rbufsize = 0
        return self.response.readline()

    def getdata(self):
        """Get the returned data parsed into a python object.  Right now
        only supports JSON encoded data.

        :return: Data is parsed as the returned data type into a python
        object.  If the data type isn't supported than the string value of
        the data is returned.
        """
        if self.error_status:
```

Copyright © 2024, Oracle and/or its affiliates

```
            return None
        data = self.body
        if data:
            content_type = self.getheader("Content-Type")
            if content_type.startswith("application/json"):
                data = json.loads(data)
        return data

    def getheader(self, name):
        """Get an HTTP header with the given name from the results

        :param name: HTTP header name
        :return: The header value or None if no value is found
        """
        if self.error_status:
            return None
        info = self.response.info()
        return info.getheader(name)

    def debug(self):
        """Get debug text containing HTTP status and headers"""
        if self.error_status:
            return repr(self.response) + "\n"

        msg = httplib.responses.get(self.status, "Unknown")
        hdr = "HTTP/1.1 %d %s\n" % (self.status, msg)
        return hdr + str(self.response.info())


class RestRunner(object):
    """REST request runner for a background client call.  Clients can obtain
    the result when it is ready by calling result()
    """
    def _init_(self, client, request, **kwargs):
        self._result = None                  # REST result from request
        self._called = threading.Condition()  # Result available condition
        self.client = client                 # Client used to run request
```

Copyright © 2024, Oracle and/or its affiliates

```
        self.request = request             # REST Request
        self.verbose = kwargs.get("verbose")

    def _str_(self):
        url = self.client.REST_URL % (self.client.host, self.request.path)
        out = "%s %s %s\n" % (self.request.method, url, self.request.data)
        if self.isdone():
            if self.verbose:
                out += self._result.debug()
                out += "\n"
            out += str(self._result)
            out += "\n"
        else:
            out += "waiting"
        return out

    def run(self):
        """Thread run routine.  Should only be called by thread"""
        try:
            result = self.client.execute(self.request)
        except Exception as err:
            result = RestResult(err, -1)
    with self._called:
            self._result = result
            self._called.notify_all()

    def isdone(self):
        """Determine if the REST call has returned data.

        :return: True if server has returned data, otherwise False
        """
        with self._called:
            return self._result is not None

    def result(self, timeout=0):
        """Get the REST call result object once the call is finished.

```

Copyright © 2024, Oracle and/or its affiliates

```
        :param timeout: The number of seconds to wait for the response to
                        finish
        :returns: RestResult or None if not finished.
        """
        with self._called:
            if self._result:
                return self._result
            else:
                self._called.wait(timeout)
                return self._result

    def cancel(self):
        if self.isdone():
            result = self.result()
            if result:
                result.fp.close()


class RestClient(object):
    """A REST Client API class to access the ZFSSA REST API"""
    REST_URL = https://%s:215%s
    ACCESS_URL = https://%s:215/api/access/v1

    def _init_(self, host, user=None, password=None, session=None):
        """Create a client that will communicate with the specified ZFSSA
        host.  If user and password are not supplied then the client must
        login before making calls.

        :param host: Appliance host name/ip address
        :param user: Management user name
        :param password: Management user password.
        :param session: Create a client using an existing session
        """
        self.host = host
        self.opener = urllib2.build_opener(urllib2.HTTPHandler)
        self.services = None
        if session:
```

Copyright © 2024, Oracle and/or its affiliates

```
            self.opener.addheaders = [
                ("X-Auth-Session", session),
                ('Content-Type', 'application/json')]
        elif user and password:
            auth = "%s:%s" % (user, password)
            basic = "Basic %s" % base64.encodestring(auth).replace('\n', '')
            self.opener.addheaders = [
                ("Authorization", basic),
                ('Content-Type', 'application/json')]

    def login(self, user, password):
        """
        Create a login session for a client.  The client will keep track of
        the login session information so additional calls can be made without
        having to supply credentials.

        :param user: The login user name
        :param password: The ZFSSA user password
        :return: The REST result of the login call
        """
        if self.services:
            self.logout()

        auth = "%s:%s" % (user, password)
        basic = "Basic %s" % base64.encodestring(auth).replace('\n', '')
        url = self.ACCESS_URL % self.host
        request = urllib2.Request(url, '')
        request.add_header('Authorization', basic)
        request.get_method = lambda: 'POST'

        try:
            result = RestResult(self.opener.open(request))
            if result.status == httplib.CREATED:
```

Copyright © 2024, Oracle and/or its affiliates

```
                session = result.getheader("X-Auth-Session")
                self.opener.addheaders = [
                    ("X-Auth-Session", session),
                    ('Content-Type', 'application/json')]
                data = result.getdata()
                self.services = data["services"]
        except urllib2.HTTPError as e:
            result = RestResult(e)
        return result

    def logout(self):
        """Logout of the appliance and clear session data"""
        request = urllib2.Request(self.ACCESS_URL % self.host)
        request.get_method = lambda: "DELETE"
        result = self.call(request)
        self.opener.addheaders = None
        self.services = None
        return result

    def _service_url(self, module, version=None):
        url = None
        for service in self.services:
            if module == service['name']:
                if version and service['version'] != version:
                    continue
                url = service['uri']
                break
        return url

    def url(self, path, **kwargs):
        """
        Get the URL of a resource path for the client.

        :param path: Resource path
        :key service: The name of the REST API service
        :key version: The version of the service
        :return:
```

Copyright © 2024, Oracle and/or its affiliates

```
        """
        service = kwargs.get("service")
        if service:
            url = self._service_url(service, kwargs.get("version")) + path
        else:
            url = self.REST_URL % (self.host, path)
        return url

    def call(self, request, background=False):
        """Make a REST API call using the specified urllib2 request"""
        if background:
            runner = RestRunner(self, request)
            thread = threading.Thread(target=runner)
            thread.start()
            return runner
        try:
            response = self.opener.open(request)
            result = RestResult(response)
        except urllib2.HTTPError as e:
            result = RestResult(e)
        return result

    def get(self, path, **kwargs):
        """Make a REST API GET call

        :param path: Resource path
        :return: RestResult
        """
        request = urllib2.Request(self.url(path, **kwargs))
        return self.call(request, kwargs.get("background"))

    def delete(self, path, **kwargs):
        """Make a REST API DELETE call

        :param path:
        :return: RestResult
        """
```

Copyright © 2024, Oracle and/or its affiliates

```
        request = urllib2.Request(self.url(path, **kwargs))
        request.get_method = lambda: "DELETE"
        return self.call(request, kwargs.get("background"))

    def put(self, path, data="", **kwargs):
        """Make a REST API PUT call

        :param path: Resource path
        :param data: JSON input data
        :return: RestResult
        """
        url = self.url(path, **kwargs)
        if not isinstance(data, (str, unicode)):
            data = json.dumps(data)
        request = urllib2.Request(url, data)
        request.get_method = lambda: "PUT"
        request.add_header('Content-Type', "application/json")
        return self.call(request, kwargs.get("background"))

    def post(self, path, data="", **kwargs):
        """Make a REST API POST call

        :param path: Resource path
        :param data: JSON input data
        :return: RestResult
        """
        url = self.url(path, **kwargs)
        if not isinstance(data, (str, unicode)):
            data = json.dumps(data)
        request = urllib2.Request(url, data)
        request.get_method = lambda: "POST"
        request.add_header('Content-Type', "application/json")
        return self.call(request, kwargs.get("background"))
    def execute(self, request, **kwargs):
        """Make an HTTP REST request

        :param method: HTTP command (GET, PUT, POST, DELETE)
```

Copyright © 2024, Oracle and/or its affiliates

```python
        :param path: Resource path
        :param data: JSON input data
        """
        if request.method.lower() == "get":
            return self.get(request.path, **kwargs)
        if request.method.lower() == "put":
            return self.put(request.path, request.data, **kwargs)
        if request.method.lower() == "post":
            return self.post(request.path, request.data, **kwargs)
        if request.method.lower() == "delete":
            return self.delete(request.path, **kwargs)
        raise Exception(
            "Invalid HTTP request '%s' "
            "(Should be one of GET, PUT, POST, DELETE)" % request.method
```

Connect with us

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com   facebook.com/oracle   twitter.com/oracle

Copyright © 2024, Oracle and/or its affiliates. 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, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.