



FluidFS Scripting Solutions Guide

Scripting with CLI, REST and PowerShell APIs

Dell Compellent FS8600 Network-Attached Storage (NAS)

Jonathan Shalev, PhD.
FluidFS System Engineering
June 2017

Revisions

Revision	Date	Description
R01	January 2015	V1.0
R02	January 2016	REST and PowerShell APIs introduced in FluidFS v5
R03	February 2017	Update CLI section for FluidFS v6
R04	June 2017	Typo fix in the ssh-keygen command



© 2015, 2016, 2017 Dell-EMC Inc. All Rights Reserved. Dell, the Dell logo, and other Dell names and marks are trademarks of Dell-EMC Inc. in the US and worldwide. All other trademarks mentioned herein are the property of their respective owners.



Table of contents

Revisions	2
1 Preface.....	5
1.1 Audience.....	5
1.2 Purpose	5
1.3 Disclaimer	5
1.4 Customer Support.....	5
2 Introduction	6
2.1 CLI Scripting Overview	6
2.2 PowerShell Scripting Overview	6
2.3 REST API Scripting Overview.....	7
3 Connecting to the FluidFS Cluster CLI through SSH without Using a Password.....	8
3.1 Setting up an ssh key-pair Using a Linux Client	8
4 CLI Scripting.....	11
4.1 CLI Scripting Examples	12
4.2 Basic CLI Commands.....	12
4.3 Parsable CLI Scripting	13
5 PowerShell Scripting.....	14
5.1 Setting Up PowerShell Scripting	14
5.2 Initializing a PowerShell Connection to Enterprise Manager.....	14
5.3 Examples of PowerShell Scripting	15
6 REST API Scripting.....	18
6.1 Setting Up REST API Scripting	19
6.2 Initializing a REST API Connection to Enterprise Manager.....	19
6.3 Guidelines for Using REST API	19
6.4 Examples of REST API Scripting	20
7 Additional Resources	24



1 Preface

1.1 Audience

This document is intended for systems, networking, or storage administrators who are responsible for the day-to-day management responsibilities of a Dell Compellent FS8600 FluidFS NAS solution.

Proper management of an FS8600 requires administrators (or teams of administrators) who can:

- Manage and configure enterprise-class Fibre Channel SAN and Ethernet networks
- Use any necessary enterprise-grade backup software
- Use the Dell Compellent Storage Center
- Perform general purpose NAS administration

1.2 Purpose

The purpose of this document is to help the storage administrator understand the FS8600 scripting mechanisms. This document is not intended to be a primer or Dell Compellent FS8600 introductory resource for any of the subject matters involved, and it assumes at least basic knowledge of many of the subjects covered in this document.

This document should be used in conjunction with other Dell Compellent resources, such as the Dell Compellent Storage Center Connectivity Guide, FS8600 Admin Guide and Hardware Manual, Enterprise Manager User Guide, or any other available documentation resources.

1.3 Disclaimer

The information contained within this document is intended to provide best practices and general recommendations only. Actual configurations in customer environments may need to vary due to individual circumstances, budget constraints, service level agreements, applicable industry-specific regulations, or other factors. Configurations should be tested before implementing them in a production environment.

1.4 Customer Support

Dell Compellent provides live support at 1-866-EZSTORE (866.397.8673), 24 hours a day, 7 days a week, 365 days a year. For additional support, email Dell Compellent at support@compellent.com. Dell Compellent responds to emails during standard business hours (US Central Time).



2 Introduction

FluidFS is an enterprise-class distributed file system that allows customers to easily and efficiently manage file data. FluidFS removes the scaling limitations of traditional file systems. It also supports scale-out performance and scale-up capacity expansion, all within a single namespace for easier administration. Because FluidFS optimizes performance and scalability, it is an excellent choice for a wide range of use cases and deployment environments.

The FS8600 NAS appliance provides a number of management interfaces. The two main interfaces are Enterprise Manager, a graphical interface, and CLI, a command line interface. Both of these interfaces are useful for ad hoc changes.

FluidFS v5 introduced two additional management APIs. One is a PowerShell API and the other is a REST API.

There is an important difference in how the APIs are accessed. Whereas the CLI is accessed with an ssh connection to a specific FluidFS system, using a FluidFS cluster administrator's credentials, both the PowerShell and REST APIs are accessed by connecting to an Enterprise Manager Data Collector (EMDC), using an EMDC administrator's credentials. For both the PowerShell and REST API cases, after connecting to an EMDC, the next step is to choose one of the FluidFS systems that is managed by the EMDC.

Some administration tasks need to be performed at regular intervals, or on demand, without requiring interaction with the FluidFS administrator. Some example of such tasks are: creating NAS volumes, creating file shares, creating quotas, or generating reporting data such as a list of volumes with their current sizes. For these cases, scripting can be useful.

This guide will demonstrate how to automate FluidFS management via scripting.

Note: This Guide is applicable to FluidFS v3 & above for the CLI interface, with the REST and PowerShell APIs applying to FluidFS v5 and above.

2.1 CLI Scripting Overview

“CLI Scripting” refers to creating a script file containing **FluidFS CLI commands** for performing a certain administration task, which can be run ad hoc or in a scheduled manner by an authorized user

To use scripts containing FluidFS CLI commands:

1. Create an appropriately named script.
2. Edit the script file and add FluidFS CLI commands that are required to perform the administration task (see *CLI Scripting Examples*).
3. Set up an SSH key pair (see *Setting up an ssh key-pair Using a Linux Client*).
4. Run the script manually whenever required.
- OR -
Schedule the script to run periodically, for example with crontab scheduling.

2.2 PowerShell Scripting Overview

“PowerShell Scripting” refers to creating a script file containing **PowerShell commands** for performing a certain administration task, which can be run ad hoc or in a scheduled manner by an authorized user.



To use scripts containing FluidFS PowerShell commands:

1. Set up a directory to be accessed from a MS Windows machine, with the appropriate PowerShell libraries.
2. Create an appropriately named script.
3. Edit the script file and add FluidFS PowerShell commands that are required to perform the administration task (see the examples in the PowerShell section below).
4. Run the script manually whenever required.
- OR -
Schedule the script to run periodically, using a windows scheduling tool.

2.3 REST API Scripting Overview

“REST API Scripting” refers to running a program using **REST API commands** using the HTTPS protocol for performing a certain administration task, which can be run ad hoc or in a scheduled manner by an authorized user.

To use the REST API commands you can use any programming language that supports the HTTPS protocol. We will demonstrate the protocol in this guide with the **curl** linux command.

1. Create an appropriately named script.
2. Edit the script file and add FluidFS REST API commands that are required to perform the administration task (see the examples in the REST API section below).
3. Run the script manually whenever required.
- OR -
Schedule the script to run periodically.



3 Connecting to the FluidFS Cluster CLI through SSH without Using a Password

To run CLI commands in a script from a client machine to FluidFS, you will need to first set up a passwordless ssh connection between the client and the FluidFS cluster by using RSA keys. This is done by generating an ssh key-pair (a public and a private RSA key) on a Linux workstation, and providing the public key to the cluster. This can be accomplished with the following steps (these steps are also listed in the Enterprise Manager Administrator's Guide).

3.1 Setting up an ssh key-pair Using a Linux Client

1. Log on to a UNIX/Linux workstation for which you want to bypass the SSH login prompt. For this example, the username used for the login is "user".
2. From the command line, type the following command:

```
ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
```

3. A random SSH key pair is generated at `~/.ssh/id_rsa.pub` (public key) and `~/.ssh/id_rsa` (private key)

```
-bash-4.1$ ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Your identification has been saved in /home/jonathan/.ssh/id_rsa.
Your public key has been saved in /home/jonathan/.ssh/id_rsa.pub.
The key fingerprint is:
f5:80:82:a2:8c:d3:17:56:58:1a:99:17:94:ca:a5:9b vcsa@linux-jonathans
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      .*=0      |
|      ++       |
|     ..=.. . o  |
| oo o+. . . o   |
| +.. .o S      |
| . .E          |
|               |
|               |
|               |
+-----+
-bash-4.1$ _
```

4. The public key (the contents of the file `~/.ssh/id_rsa.pub`) must now be provided to the FluidFS cluster. Copy the SSH public key by displaying it (with the `cat` Linux command) and selecting it:

```
[user@jonathan-linux2 ~]$ cat /home/user/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAABIWAAAQEAuMCwge5HWSYakFlarpPjmVw6yHhSqSP9TWjoP6Z76ZIsfY5I8gLS+ngk82iif5oij+RmAB
w5PT0lnZohnUmmZMo1fytSIXNpt38QLXv9pjZICs3TNRk9T9WpvmUjJYlv1uT8N5K+Se06KlnyXFJGey28g74CFIYYJdtXx5GAX0X3j/Pa
RR8lBGwySxd3oa/vUpfpmLLES+AKEAuaeLDfFAaEvcqgrXkyzY+HG3JqsZMMjVhcKo3UyVRmFYgIBaie5fEQ8BVQtQ4rK7ZPE7oiew8g7
1w6ZMEuti0iVXxpQzwFqqZmAF1Klp20gbbgNfZAND2Pr7LGmqkebnToPIBoQ== user@jonathan-linux2
[user@jonathan-linux2 ~]$
```



5. Log on to the FluidFS CLI using a password:

```
(jonathans) vncsrv3:4995 ~ => ssh cli@cplsup3
Last login: Sun Feb  5 17:04:27 2017 from 10.48.28.63

Welcome to "cplsup3" (6.0.003162)
Installed on Sun Jan  8 11:32:42 CST 2017

login as: Administrator
Administrator's password: *****
Welcome to CLI, type 'help' for more information.

Default commands:

    back
    main
    history
    exit
    quit
    help
    find

Available menus:

    multitенancy
    NAS-pool
    environment
    maintenance
    hardware
    events

CLI> █
```

6. Add an SSH user name to the administrator you would like to use with the following command:



```
CLI/maintenance/administrators> list
```

Domain	UserName	Email	IsGlobal	SSHUserName	SSHKeys		
cplsup3	Administrator		Yes	Administrator	<table><tr><td>MachineName</td><td>SSHKey</td></tr></table>	MachineName	SSHKey
MachineName	SSHKey						
cplsup3	jonathan		Yes		<table><tr><td>MachineName</td><td>SSHKey</td></tr></table>	MachineName	SSHKey
MachineName	SSHKey						

```
CLI/maintenance/administrators> edit
```

```
Administrator jonathan
```

```
CLI/maintenance/administrators> edit cplsup3 jonathan -SSHUserName jonathan
```

```
CLI/maintenance/administrators> list
```

Domain	UserName	Email	IsGlobal	SSHUserName	SSHKeys		
cplsup3	Administrator		Yes	Administrator	<table><tr><td>MachineName</td><td>SSHKey</td></tr></table>	MachineName	SSHKey
MachineName	SSHKey						
cplsup3	jonathan		Yes	jonathan	<table><tr><td>MachineName</td><td>SSHKey</td></tr></table>	MachineName	SSHKey
MachineName	SSHKey						

```
CLI/maintenance/administrators>
```

7. Type the following command to provide the public key to the FluidFS cluster (<SSH_key> should be replaced by pasting the characters you copied in step 4, enclosed in double quotation marks):

maintenance administrators passwordless-access add-ssh-keys <domain-name> <admin-name> <machine-name> -SSHKey "<SSH_key>"

```
CLI> maintenance administrators passwordless-access add-ssh-keys cplsup3 jonathan vncsrv3 "ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAq5sH0FxfvuGbh4MR6FNjBTAfUQ5kIF5VrMAT6lllB0EUTEbMeTo5QMZXoWiaWTIF7q/zj+ytT0bD28X0tlUNDAYnylZA9kZrk2Jl3wC7PBksKD0m6L/mrt6wn5llyKknmKGpAPuu8oqAvWN0x7l2tCGL036ph1r4WSm5L1rNTTAvZ0YNY3dCsMcWgeCPHvsrKEzXpo0WU3rgh53UUAdxNCQkQc5wNCGJdqm7jDFuhOuYpV3nJIWdqGLj1vBFH+dM2zEnBro5JdyzcIPF6cpYE7Ndv76bs6EP6Dx/exgipjhcI0xlvjdwaNfXWl7yu2m23SG4tCwWS2dHXf6FFvxQ== jonathans@vncsrv3"
```

8. You have now completed the authentication setup, and can run CLI commands (or enter the CLI interactively) from the Linux workstation without using a password.

CLI Scripting

After performing step 3.1, you can use the following command to log on to the FluidFS cluster from the workstation without needing a password:

ssh <FluidFS_administrator_user_name>@<FluidFS_client_VIP_or_name>

```
(jonathans) vncsrv3:5000 ~ => ssh jonathan@cplsup3
Last login: Mon Feb  6 11:41:27 2017 from 10.48.28.64
```

```
Welcome to "cplsup3" (6.0.003162)
Installed on Sun Jan  8 11:32:42 CST 2017
```

Welcome to CLI, type 'help' for more information.

Default commands:

```
back
main
history
exit
quit
help
find
```

Available menus:

```
multitenancy
NAS-pool
environment
maintenance
hardware
events
```

CLI> █

You can also use the following format to run commands from the

workstation without needing a password:

ssh <FluidFS_administrator_user_name>@<FluidFS_client_VIP_or_name> <CLI_command>

```
[user@jonathan-linux2 ~]$ ssh Administrator@cplsup3 hardware NAS-appliances list
```

Appliance ID	Appliance Service Tag	Is File System Member	Controllers						
1	7CC5J5J	Yes	<table border="1"> <thead> <tr> <th>Controller ID</th> <th>Cluster Member</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Yes</td> </tr> <tr> <td>1</td> <td>Yes</td> </tr> </tbody> </table>	Controller ID	Cluster Member	0	Yes	1	Yes
Controller ID	Cluster Member								
0	Yes								
1	Yes								

```
[user@jonathan-linux2 ~]$ █
```

4.1 CLI Scripting Examples

This section contains examples of scripting.

4.2 Basic CLI Commands

Note: The examples here are of the actual ssh commands that would be embedded in the scripts.

The first example lists the client subnets defined on the machine:

ERROR: unrecognized token: 'networking'

```
(jonathans) vncsrv3:5002 ~ => ssh jonathan@cplsup3 environment network subnets list
```

Network Id	Prefix Length	VLAN Tag	Private IPs
10.140.84.0	27	3178	10.140.84.8,10.140.84.9
172.41.0.0	16	0	172.41.2.220,172.41.2.221

```
(jonathans) vncsrv3:5003 ~ => █
```

The second example lists the NAS volumes and their sizes in the cplsup3 tenant:

```
(jonathans) vncsrv3:5005 ~ => ssh jonathan@cplsup3 multitenanty tenants cplsup3 NAS-Volumes list-volumes space
```

Name	Size	Used Space	Unused Space	Over Committed Space	Space Provisioning	Unused Reserved Space	Clone
GabbySP	1.0 TB	1.00 GB	1023.00 GB	0.0 MB	Thin	0.0 MB	No
Varonis	500.0 GB	163.28 MB	499.84 GB	0.0 MB	Thin	0.0 MB	No
hdvol	50.0 GB	0.41 MB	50.00 GB	0.0 MB	Thin	0.0 MB	No
jvol1	800.0 GB	27.37 GB	772.63 GB	0.0 MB	Thin	0.0 MB	No
jvol1-rep-from-xiosup1	1.5 TB	1.00 TB	475.58 GB	0.0 MB	Thin	0.0 MB	No
nimrod	500.0 GB	4.35 GB	495.65 GB	0.0 MB	Thin	0.0 MB	No



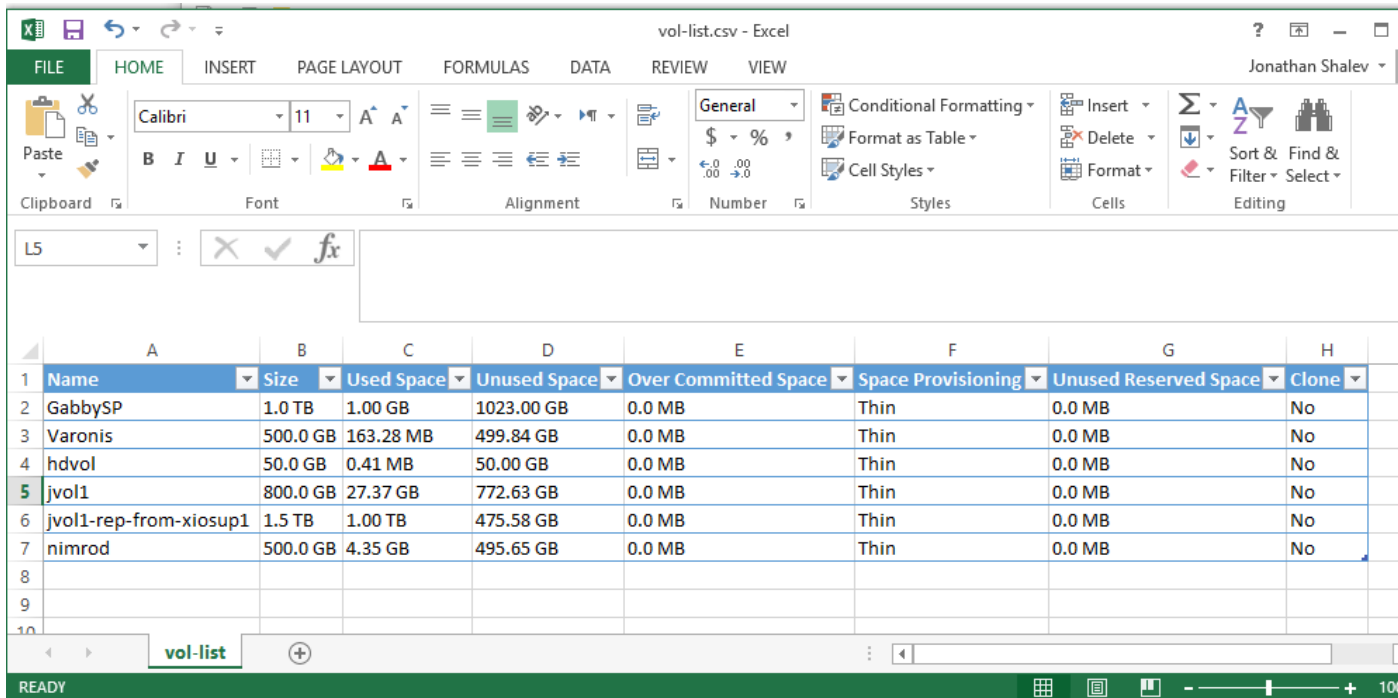
4.3 Parsable CLI Scripting

Many CLI listing commands accept a `--csv` parameter. Providing this parameter causes the CLI command to produce its output in a comma-separated-values format, that can be easily parsed by a program, or even opened with Excel or other spreadsheet applications.

For example, here is a list of the NAS volumes in tenant `cplsup3` and their sizes in this format:

```
(jonathans) vncsrv3:5006 ~ => ssh jonathan@cplsup3 multitenancy tenants cplsup3 NAS-Volumes list-volumes space --csv
Name,Size,Used Space,Unused Space,Over Committed Space,Space Provisioning,Unused Reserved Space,Clone
GabbySP,1.0 TB,1.00 GB,1023.00 GB,0.0 MB,Thin,0.0 MB,No
Varonis,500.0 GB,163.28 MB,499.84 GB,0.0 MB,Thin,0.0 MB,No
hdvol,50.0 GB,0.41 MB,50.00 GB,0.0 MB,Thin,0.0 MB,No
jvol1,800.0 GB,27.37 GB,772.63 GB,0.0 MB,Thin,0.0 MB,No
jvol1-rep-from-xiosup1,1.5 TB,1.00 TB,475.58 GB,0.0 MB,Thin,0.0 MB,No
nimrod,500.0 GB,4.35 GB,495.65 GB,0.0 MB,Thin,0.0 MB,No
(jonathans) vncsrv3:5007 ~ =>
```

Copying the command's output into a CSV file and opening it in Excel produces the following:



	A	B	C	D	E	F	G	H
1	Name	Size	Used Space	Unused Space	Over Committed Space	Space Provisioning	Unused Reserved Space	Clone
2	GabbySP	1.0 TB	1.00 GB	1023.00 GB	0.0 MB	Thin	0.0 MB	No
3	Varonis	500.0 GB	163.28 MB	499.84 GB	0.0 MB	Thin	0.0 MB	No
4	hdvol	50.0 GB	0.41 MB	50.00 GB	0.0 MB	Thin	0.0 MB	No
5	jvol1	800.0 GB	27.37 GB	772.63 GB	0.0 MB	Thin	0.0 MB	No
6	jvol1-rep-from-xiosup1	1.5 TB	1.00 TB	475.58 GB	0.0 MB	Thin	0.0 MB	No
7	nimrod	500.0 GB	4.35 GB	495.65 GB	0.0 MB	Thin	0.0 MB	No
8								
9								
10								

5 PowerShell Scripting

5.1 Setting Up PowerShell Scripting

The PowerShell SDK package can be downloaded from the Compellent Knowledge Center or from the SCv20x0 area of the Dell Support site.

Once you have the set of files, place them in a directory accessible from your Windows computer. We refer to this directory as the base directory and the Windows machine as the scripting machine.

All FluidFS PowerShell scripting is performed via an instance of Enterprise Manager Data Collector (EMDC). The EMDC must be accessible from the scripting machine, and you should have credentials (user name and password) of an EMDC administrator.

5.2 Initializing a PowerShell Connection to Enterprise Manager

Before starting, one must import the Dell Storage PowerShell module and connect to the EMDC.

Start Powershell, change the directory to the base directory.

To import the module, run the following command:

```
PS> Import-Module DellStorage.ApiCommandSet.psd1
```

We recommend not having the password displayed or saved in plain text. To request that the password be obtained from the user without it being displayed, run the following and enter the password when prompted:

```
PS > $pass= Read-Host -AsSecureString -Prompt "Please provide password"
```

To connect to EMDC, run the following command:

```
PS > $conn = Connect-DellApiConnection -Host <EMDC-IP-Address> -Port 3033 -User  
<EMDC-admin-username> -password $pass
```

Port 3033 is the port of the default EMDC connection. If you changed the default port, you should use the port number you changed to.

Now that you have the connection object assigned to the variable **\$conn**, you can use it to retrieve information from FluidFS clusters and also to run general management commands on them.



Here is a screenshot of the initialization performed on an EMDC that manages two FluidFS clusters.

```
PS > Import-Module .\DellStorage.ApiCommandSet.psdl
PS > $pass = Read-Host -AsSecureString -Prompt "Please provide EMDC password for Administrator: "
Please provide EMDC password for Administrator: : *****
PS > $conn = Connect-DellApiConnection -Host 172.41.200.98 -Port 3033 -User Administrator -password $pass
PS > $conn

Connection                               Connected BaseConnection
-----
ApiConnection                           True ApiConnection
```

5.3 Examples of PowerShell Scripting

There could be multiple clusters managed in the same instance of EMDC. In general, the first step will be to get the “InstanceId” of the cluster we would like to work with. This InstanceId is a long string of hexadecimal characters, such as **InstanceId : f5d9d4cc-f8c1-4175-a12d-37e0f3c5bf6b**.

What we usually know is the cluster name. Here is how we get from the cluster name to the InstanceId.

First, we get a list of the FluidFS clusters managed by the EMDC. Recall that the \$conn variable contains our connection to the EMDC.

```
PS > $clusters=Get-DellFluidFsCluster -Connection $conn
PS > $clusters

ClusterId      : b03a0ba7-c332-487f-82fc-8ef430ea38cc
Name           : cplsup7
UniqueName     : FluidFsCluster_b03a0ba7-c332-487f-82fc-8ef430ea38cc
InstanceId     : b03a0ba7-c332-487f-82fc-8ef430ea38cc
InstanceName   : cplsup7
ObjectType     : FluidFsCluster

ClusterId      : f5d9d4cc-f8c1-4175-a12d-37e0f3c5bf6b
Name           : cplsup3
UniqueName     : FluidFsCluster_f5d9d4cc-f8c1-4175-a12d-37e0f3c5bf6b
InstanceId     : f5d9d4cc-f8c1-4175-a12d-37e0f3c5bf6b
InstanceName   : cplsup3
ObjectType     : FluidFsCluster

PS >
```

We used the Get-DellFluidFsCluster command, with the \$conn parameter, to get a list of FluidFS clusters managed by our EMDC into the \$clusters PowerShell variable.

There are two clusters, cplsup7 with an instanceId ending with 38cc, and cplsup3, with an InstanceId ending with bf6b.

We can use PowerShell filters to choose the instanceId according to the name. Here is an example getting the InstanceId of the cluster called cplsup7:

```
PS > $myCluster = $clusters | ? {$_.name -eq "cplsup7"}
PS > $myCluster

ClusterId      : b03a0ba7-c332-487f-82fc-8ef430ea38cc
Name           : cplsup7
UniqueName     : FluidFsCluster_b03a0ba7-c332-487f-82fc-8ef430ea38cc
InstanceId     : b03a0ba7-c332-487f-82fc-8ef430ea38cc
InstanceName   : cplsup7
ObjectType     : FluidFsCluster

PS > $myInstanceId = $myCluster.InstanceId
PS > $myInstanceId
b03a0ba7-c332-487f-82fc-8ef430ea38cc
PS >
```

Now that we have the InstanceId for the cluster, we can run commands on this cluster.

Here is an example to get a list of volumes on the cluster, then to print the names of the volumes, and finally, for each volume, to report how much space is used and the size of the volume:

```
PS > $volumeList = Get-DellFluidFsClusterNasVolumeListAssociation -Connection $conn -Instance $myInstanceId
PS > $volumeList | foreach { $_.Name }
jvol1
jvolt
jvol1-rep
PS > $volumeList | foreach { "Volume {0,-10} has used {1,-14} of {2,-14}" -f $_.Name, $_.UsedSpace, $_.Size.ToString() }
Volume jvol1      has used 2.65 GB      of 500 GB
Volume jvolt      has used 296.5 KB    of 500 GB
Volume jvol1-rep  has used 2.65 GB      of 500 GB
PS >
```

The next example gives a report on the SMB shares on our cluster. For each share, we print the name of the NAS volume and the path to the share in the NAS volume:

```
PS > $shareList = Get-DellFluidFsClusterSmbShareListAssociation -Connection $conn -Instance $myInstanceId
PS > $shareList | foreach { "SMB Share {0,-10}: Volume: {1,-10}, Path: {2,-15}" -f $_.ShareName, $_.VolumeName, $_.Path }
SMB Share jshare1 : Volume: jvol1      , Path: /
SMB Share sports  : Volume: jvol1      , Path: /sportsFolder
SMB Share Finance : Volume: jvol1      , Path: /FinanceFolder
SMB Share TopOfJvolt: Volume: jvolt      , Path: /
SMB Share RepShare : Volume: jvol1-rep    , Path: /
PS >
```

This covers examples of reporting. Now we show how to create objects. We will create a new NAS volume called Weather and repeat the NAS volume query we showed previously and see the new NAS volume in the list.




```

PS > $newvol = New-DellFluidFsNasVolume -Connection $conn -ClusterId $myInstanceId -Name Weather -Size 100000000
PS > $volumelist = Get-DellFluidFsClusterNasVolumeListAssociation -Connection $conn -Instance $myInstanceId
PS > $volumelist | foreach { "Volume {0,-10} has used {1,-14} of {2,-14}" -f $_.Name, $_.UsedSpace, $_.Size.ToString() }

Volume jvol1      has used 2.65 GB      of 500 GB
Volume jvolt      has used 296.5 KB     of 500 GB
Volume jvol1-rep  has used 2.65 GB      of 500 GB
Volume Weather    has used 20 KB        of 95 MB
PS >

```

As a final example of using PowerShell to create an object in Fluidfs, in this new NAS volume, we create a new Share to the root folder of the volume, also called Weather. Note that we use the \$newvol variable that was returned when we created the new NAS volume. After creating the share, we repeat the list-shares command and see the new share.

```

PS > $newShare = New-DellFluidFsSmbShare -ClusterId $myInstanceId -NasVolumeId $newvol.NasVolumeId -Path / -ShareName Forecasts
PS > $sharelist = Get-DellFluidFsClusterSmbShareListAssociation -Connection $conn -Instance $myInstanceId
PS > $sharelist | foreach { "SMB Share {0,-10}: Volume: {1,-10}, Path: {2,-15}" -f $_.ShareName, $_.VolumeName, $_.Path }

SMB Share jshare1 : Volume: jvol1      , Path: /
SMB Share sports  : Volume: jvol1      , Path: /sportsFolder
SMB Share Finance : Volume: jvol1      , Path: /FinanceFolder
SMB Share TopOfJvolt: Volume: jvolt      , Path: /
SMB Share RepShare : Volume: jvol1-rep , Path: /
SMB Share Forecasts : Volume: Weather    , Path: /
PS >

```

This concludes the PowerShell examples – enumerating clusters, choosing the required cluster and getting its unique ID, listing NAS volumes and SMB shares on the required cluster, and creating a new NAS volume and a new SMB share

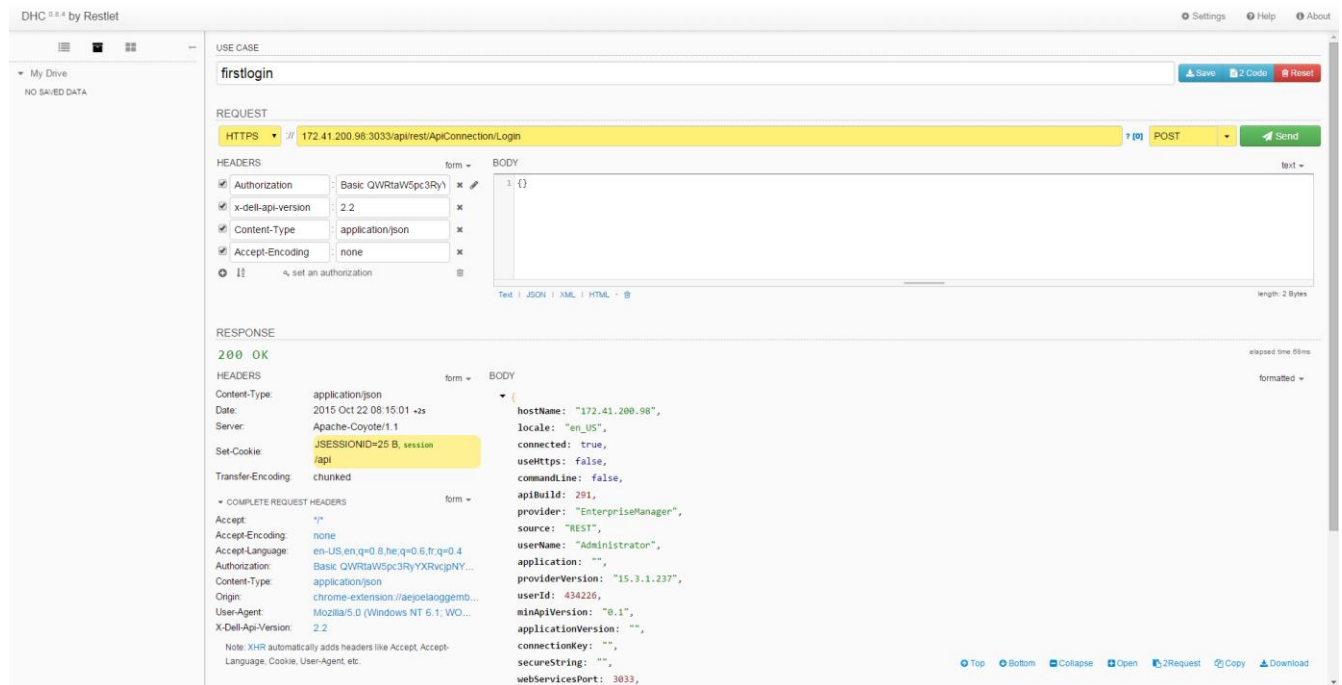


6 REST API Scripting

REST API, a web-based protocol, can be accessed in many ways, similar to the way that any web page can be accessed from different operating systems, browsers, applications, robots, etc. This is because running RESTful requests is performed by submitting an HTTPS URI to the EMDC web server.

In this guide, I will be using the linux curl (transfer a URL) command, as it is completely text based and will be easy to demonstrate. There are many other possibilities, including every programming language that includes a web programming library. You can use the graphical DHC (Dev HTTP Client) plugin for Chrome to play around with it too.

This is what the DHC screen looks like after performing a login (you don't need to read the small print):



This is how I will be showing the same thing with curl:

```
(jonathans) vncsrv3:3236 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/ApiConnection/Login -X POST -d '{"hostname": "172.41.200.98", "locale": "en_US", "connected": true, "useHttps": false, "commandLine": false, "apiBuild": 291, "provider": "EnterpriseManager", "source": "REST", "userName": "Administrator", "application": "", "providerVersion": "15.3.1.237", "userId": 434226, "minApiVersion": "0.1", "applicationVersion": "", "connectionKey": "", "secureString": "", "webServicesPort": 3033, "apiVersion": "2.2", "sessionKey": "1444629931186", "objectType": "ApiConnection", "instanceId": "0", "instanceName": "ApiConnection"}' -H 'content-type: application/json' -H 'x-dell-api-version: 2.2'
```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=aVf9P4sEFd6mGdJu-MZiDrMV; Path=/api; Secure
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 22 Oct 2015 05:19:15 GMT

```
{
  "hostname": "172.41.200.98",
  "locale": "en_US",
  "connected": true,
  "useHttps": false,
  "commandLine": false,
  "apiBuild": 291,
  "provider": "EnterpriseManager",
  "source": "REST",
  "userName": "Administrator",
  "application": "",
  "providerVersion": "15.3.1.237",
  "userId": 434226,
  "minApiVersion": "0.1",
  "applicationVersion": "",
  "connectionKey": "",
  "secureString": "",
  "webServicesPort": 3033,
  "apiVersion": "2.2",
  "sessionKey": "1444629931186",
  "objectType": "ApiConnection",
  "instanceId": "0",
  "instanceName": "ApiConnection"
}
```

(jonathans) vncsrv3:3236 ~/SCRIPTS/REST-API =>



6.1 Setting Up REST API Scripting

All REST API scripting is performed via an instance of Enterprise Manager Data Collector (EMDC). The EMDC must be accessible from the scripting machine, and you should have credentials (user name and password) of an EMDC administrator. If you can access <https://<EMDC-IP-address>:3033/api/rest> (we will call this the **EMDC API URL**) then you are good to go. Note that only secure (https) connections are allowed.

Documentation for the REST API is part of the Enterprise Manager SDK available from the Compellent Knowledge Center. It is provided as an HTML file – DellStorageRestApi.html.

6.2 Initializing a REST API Connection to Enterprise Manager

To connect to the Dell Storage API service, use the APICheck/LogIn endpoint with EMDC Administrator credentials. If this succeeds, it returns a JSESSION-ID cookie which must be provided for future accesses (until it expires, when a new cookie must be obtained).

You need to perform a POST request to the EMDC API URL with the APICheck/LogIn endpoint, and three headers:

- Header 1: 'x-dell-api-version: 2.2' (this header, with the version you are using, should appear in every request)
- Header 2: 'content-type: application/json'
- Header 3: Authorization – this is a basic authorization header in the format 'Basic {Base-64 encoding of Username:Password}'. It is automatically generated by curl when using the '-u Username:Password' parameter.

If the login is successful, the response will include a line with a JSESSIONID cookie, which is used for subsequent calls.

```
(jonathans) vncsrv3:3236 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/ApiConnection/Login -X POST -d '{}' -u Administrator:Manager11 -H 'content-type: application/json' -H 'x-dell-api-version: 2.2'
```

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=aVf9P4sEFd6mGdJu-MZiDrMV; Path=/api; Secure
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 22 Oct 2015 05:19:15 GMT

```
{ "hostName": "172.41.200.98", "locale": "en_US", "connected": true, "useHttps": false, "commandLine": false, "apiBuild": 291, "provider": "EnterpriseManager", "source": "REST", "userName": "Administrator", "application": "", "providerVersion": "15.3.1.237", "userId": 434226, "minApiVersion": "0.1", "applicationVersion": "", "connectionKey": "", "secureString": "", "webServicesPort": 3033, "apiVersion": "2.2", "sessionKey": 1444629931186, "objectType": "ApiConnection", "instanceId": "0", "instanceName": "ApiConnection" } (jonathans) vncsrv3:3236 ~/SCRIPTS/REST-API =>
```

This can be scripted using your favorite programming language (this section of the document is intended to be applied by programmers) to extract the cookie. Here is an example using grep and awk to get the cookie into the \$cookie shell variable (we will use this variable in the following calls):

```
(jonathans) vncsrv3:3243 ~/SCRIPTS/REST-API => cookie=`curl -k -i https://172.41.200.98:3033/api/rest/ApiConnection/Login -X POST -d '{}' -u Administrator:Manager11 -H 'content-type: application/json' -H 'x-dell-api-version: 2.2' 2>curlerr | grep Cookie | awk '{print $2}'`  
(jonathans) vncsrv3:3244 ~/SCRIPTS/REST-API => echo "Cookie is \"$cookie\""  
Cookie is "JSESSIONID=bextG9ftNMtzcU-nAy2jYX6K;"
```

6.3 Guidelines for Using REST API

After logging in, you provide another header, 'cookie: <cookie>' on subsequent requests, until it expires, at which time you should login again and request a new cookie.



Any PUT, POST, or DELETE request requires an additional HTTP header.

- Content-Type: Specifies the format of the body, must be "application/json" or "application/xml"

There some other optional headers:

- Accept: Specifies the format of the response body (JSON by default), must be "application/json" or "application/xml".
- Accept-Encoding: Specifies if the response body should be compressed, must be "gzip".
- async: This header value must be "true" to execute the request method asynchronously. If this option is used (it is available for some long-running requests), the response includes a URI to retrieve updated state information.

More complex requests, including filtering, sorting and pagination, are described in the documentation. IN the next section are some examples of common reports and actions.

6.4 Examples of REST API Scripting

There could be multiple clusters managed in the same instance of EMDC. In general, the first step will be to get the "instanceId" of the cluster we would like to work with. This instanceId is a long string of hexadecimal characters, such as "instanceId": "b03a0ba7-c332-487f-82fc-8ef430ea38cc".

What we usually know is the cluster name. Here is an example of **getting the instanceId when we know the cluster name**. We use the FluidFs/FluidFsCluster endpoint to get a list of clusters. Using the grep command to filter out the headings and the python json.tool library to format the JSON output prettily, we get the following:

```
(jonathans) vncsrv3:3252 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsCluster -X GET -d '{}'.  
H "cookie: $cookie" -H 'x-dell-api-version: 2.2' 2>/dev/null | grep '{' | python -m json.tool  
[  
  {  
    "clusterId": "b03a0ba7-c332-487f-82fc-8ef430ea38cc",  
    "instanceId": "b03a0ba7-c332-487f-82fc-8ef430ea38cc",  
    "instanceName": "cplsup7",  
    "name": "cplsup7",  
    "objectType": "FluidFsCluster"  
  },  
  {  
    "clusterId": "b29a8acb-0fdf-44c4-8bca-01ab71a5ee5b",  
    "instanceId": "b29a8acb-0fdf-44c4-8bca-01ab71a5ee5b",  
    "instanceName": "CPLsup4",  
    "name": "CPLsup4",  
    "objectType": "FluidFsCluster"  
  }  
]
```

You can analyze the JSON output and choose the instanceId that matches the cluster name you want to work with. If you prefer to work with XML output, here is the same example, **requesting the output in XML**, and using the xmllint program to pretty-print the output.



```
(jonathans) vncsrv3:3256 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsCluster -X GET -d '{}' -H "cookie: $cookie" -H 'x-dell-api-version: 2.2' -H 'Accept: application/xml' 2>/dev/null | grep '<' | xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<returnList>
  <FluidFsCluster>
    <instanceId>b03a0ba7-c332-487f-82fc-8ef430ea38cc</instanceId>
    <instanceName>cplsup7</instanceName>
    <objectType>FluidFsCluster</objectType>
    <clusterId>b03a0ba7-c332-487f-82fc-8ef430ea38cc</clusterId>
    <name>cplsup7</name>
  </FluidFsCluster>
  <FluidFsCluster>
    <instanceId>b29a8acb-0fdf-44c4-8bca-01ab71a5ee5b</instanceId>
    <instanceName>CPLsup4</instanceName>
    <objectType>FluidFsCluster</objectType>
    <clusterId>b29a8acb-0fdf-44c4-8bca-01ab71a5ee5b</clusterId>
    <name>CPLsup4</name>
  </FluidFsCluster>
</returnList>
```

Even though the correct way to handle the output is to analyze the JSON or XML result with appropriate library routines or utilities, for the examples here I will use `grep`, `awk` and `sed` commands to extract the `instanceId` for cluster `cplsup7` and put it in the `$instanceId` shell variable:

```
(jonathans) vncsrv3:3262 ~/SCRIPTS/REST-API => instanceId=$(curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsCluster -X GET -d '{}' -H "cookie: $cookie" -H 'x-dell-api-version: 2.2' 2>/dev/null | grep '<' | python -m json.tool | egrep -B1 'instanceName.*cplsup7' | head -1 | awk '{print $2}' | sed 's/"/'g' | sed 's/,/,'g')
(jonathans) vncsrv3:3263 ~/SCRIPTS/REST-API => echo $instanceId
b03a0ba7-c332-487f-82fc-8ef430ea38cc
```

Now we are ready to do some work. The first will be to **list the NAS volumes in the cluster**. Note that I used the `instanceId` variable as part of the URL, just before the “`NasVolumeList`” command. I extracted the name, sized and used space with `grep` from the formatted JSON output.

```
(jonathans) vncsrv3:3275 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsCluster/${instanceId}/NasVolumeList -X GET -d '{}' -H "cookie: $cookie" -H 'x-dell-api-version: 2.2' 2>x | grep '{' | python -m json.tool | egrep '"name|"size|"usedSpace"'
{
  "name": "jv011",
  "size": "5.36870912E11 Bytes",
  "usedSpace": "2.84918016E9 Bytes",
  "name": "jv012",
  "size": "5.36870912E11 Bytes",
  "usedSpace": "303616.0 Bytes",
  "name": "jv011-rep",
  "size": "5.36870912E11 Bytes",
  "usedSpace": "2.8482048E9 Bytes",
  "name": "Weather",
  "size": "9.961472E7 Bytes",
  "usedSpace": "303616.0 Bytes",
  "name": "Ohad",
  "size": "5.36870912E11 Bytes",
  "usedSpace": "303616.0 Bytes",
}
(jonathans) vncsrv3:3276 ~/SCRIPTS/REST-API =>
```

Similarly we can produce **a list of the SMB shares on the cluster**, printing the share name, the volume name and the path.



```
(jonathans) vncsrv3:3298 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsCluster/${instanceid}/SmbShareList -X GET -d '{}' -H "cookie: $cookie" -H 'x-dell-api-version: 2.2' 2>x | grep '{' | python -m json.tool | egrep '"shareName|path|volumeName"'
{"path": "/",
 "shareName": "jshare1",
 "volumeName": "jvoll",
 "path": "/sportsFolder",
 "shareName": "sports",
 "volumeName": "jvoll",
 "path": "/FinanceFolder",
 "shareName": "Finance",
 "volumeName": "jvoll",
 "path": "/",
 "shareName": "TopOfJvolt",
 "volumeName": "jvolt",
 "path": "/",
 "shareName": "RepShare",
 "volumeName": "jvoll-rep",
 "path": "/",
 "shareName": "Forecasts",
 "volumeName": "Weather",
 "path": "/",
 "shareName": "ohad",
 "volumeName": "Ohad"}
(jonathans) vncsrv3:3299 ~/SCRIPTS/REST-API => █
```

This covers examples of reporting. Now we show how to create objects. We will create a new NAS volume called Sports.

```
(jonathans) vncsrv3:3367 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsNasVolume -X POST -H "cookie: $cookie" -H 'Content-Type: application/json' -H 'x-dell-api-version: 2.2' -d '{"Name": "Sports", "Size": "50000", "ClusterId": "${instanceid}" }' 2>x | grep '{' | python -m json.tool | egrep '"size"|"name"'
{"name": "Sports",
 "size": "2.5165824E7 Bytes",
 "usedSpace": "303616.0 Bytes"}
(jonathans) vncsrv3:3368 ~/SCRIPTS/REST-API => █
```

Note that the size was specified in 512-byte blocks.

Next, we repeat the NAS volume query we showed previously and see the new NAS volume in the listname

```
(jonathans) vncsrv3:3371 ~/SCRIPTS/REST-API => curl -k -i https://172.41.200.98:3033/api/rest/FluidFs/FluidFsCluster/${instanceid}/NasVolumeList -X GET -d '{}' -H "cookie: $cookie" -H 'x-dell-api-version: 2.2' 2>x | grep '{' | python -m json.tool | egrep '"name"|"size"|"usedSpace"'
{"name": "jvoll",
 "size": "5.36870912E11 Bytes",
 "usedSpace": "2.858302464E9 Bytes",
 "name": "jvolt",
 "size": "5.36870912E11 Bytes",
 "usedSpace": "303616.0 Bytes",
 "name": "jvoll-rep",
 "size": "5.36870912E11 Bytes",
 "usedSpace": "2.857746944E9 Bytes",
 "name": "Weather",
 "size": "9.961472E7 Bytes",
 "usedSpace": "303616.0 Bytes",
 "name": "Ohad",
 "size": "5.36870912E11 Bytes",
 "usedSpace": "3921408.0 Bytes",
 "name": "ohadtest",
 "size": "5.36870912E11 Bytes",
 "usedSpace": "4812800.0 Bytes",
 "name": "Sports",
 "size": "2.5165824E7 Bytes",
 "usedSpace": "303616.0 Bytes"}
(jonathans) vncsrv3:3372 ~/SCRIPTS/REST-API => █
```

For our last example, we create a new SMB share called sports at the top of the Sports NAS volume

7 Additional Resources

The FluidFS section at DellTechCenter contains additional technical content such as white papers, best practices, product demos and more:

<http://en.community.dell.com/techcenter/storage/w/wiki/6935.dell-fluidfs-nas>

