PURESTORAGE

Quick Start Guide for the Pure Storage PowerShell Software Development Kit

18 July 2015

Contents

Executive Summary

We started supporting automation and management of the Pure Storage FlashArray with Windows PowerShell almost a year ago with the Pure Storage PowerShell Toolkit. The toolkit was one of the first open source projects released through GitHub. That initial release had but 7 cmdlets that provided the basics to connect, create volumes and take snapshots. Over time the Toolkit grew to support more than a 150 cmdlets that provided scripting for all operational tasks on the FlashArray.

The adoption of the Toolkit was a great indicator for us to take the next evolutional step and create a Cmdlet CommandType vs the Toolkit's approach of the Function CommandType. The newest release of PowerShell support provides 216 cmdlets that map to all of our implemented REST APIs.

The PowerShell integration to date ranges from simple FlashArray automation & management, 3rd party application integration, tools for reporting and self-management portals.

Goals and Objectives

The objective of this quick start guide is to provide step-by-step instructions in the following areas:

- Installation of the Pure Storage PowerShell SDK
- Verifying Windows PowerShell requirements
- Connecting to the Pure Storage FlashArray
- Creating Volumes
- Creating Host, Host Group and connecting Volumes
- Creating FlashRecover Snapshots
- Creating Protection Groups and adding a Host, Host Group and Volume
- Retrieving metrics

At the end of this guide you should be able to start scripting more complex solutions that enable automation and management of the Pure Storage FlashArray and any application workloads.

Audience

This document is intended for Database Administrators (DBAs), Storage Administrators, System Administrators and Windows PowerShell enthusiasts that wants to learn how to automate and manage the Pure Storage FlashArray family of products. This is a technical document that assumes a working knowledge of Windows PowerShell and RESTful APIs. These are not prerequisites to reviewing this document but are highly recommended to understand all of the different components.

Pure Storage Introduction

Pure Storage is the leading all-flash enterprise array vendor, committed to enabling companies of all sizes to transform their businesses with flash.

Built on 100% consumer-grade MLC flash, Pure Storage FlashArray delivers all-flash enterprise storage that is 10X faster, more space and power efficient, more reliable, and infinitely simpler, and yet typically costs less than traditional performance disk arrays.

Figure 1. Pure Storage FlashArray 400 and FlashArray//m.

The Pure Storage FlashArray is ideal for:

Accelerating Databases and Applications Speed transactions by 10x with consistent low latency, enable online data analytics across wide datasets, and mix production, analytics, dev/test, and backup workloads without fear.

Virtualizing and Consolidating Workloads Easily accommodate the most IO-hungry Tier 1 workloads, increase consolidation rates (thereby reducing servers), simplify VI administration, and accelerate common administrative tasks.

Delivering the Ultimate Virtual Desktop Experience Support demanding users with better performance than physical desktops, scale without disruption from pilot to >1000's of users, and experience all-flash performance for under $100/desktop.

Protecting and Recovering Vital Data Assets Provide an always-on protection for business-critical data, maintain performance even under failure conditions, and recover instantly with FlashRecover.

Pure Storage FlashArray sets the benchmark for all-flash enterprise storage arrays. It delivers:

Consistent Performance FlashArray delivers consistent <1ms average latency. Performance is optimized for the real-world applications workloads that are dominated by I/O sizes of 32K or larger vs. 4K/8K hero performance benchmarks. Full performance is maintained even under failures/updates.

Less Cost than Disk Inline de-duplication and compression deliver 5 – 10x space savings across a broad set of I/O workloads including Databases, Virtual Machines and Virtual Desktop Infrastructure.

Mission-Critical Resiliency FlashArray delivers >99.999% proven availability, as measured across the Pure Storage installed base and does so with non-disruptive everything without performance impact.

Disaster Recovery Built-In FlashArray offers native, fully-integrated, data reduction-optimized backup and disaster recovery at no additional cost. Setup disaster recovery with policy-based automation within minutes. And, recover instantly from local, space-efficient snapshots or remote replicas.

Simplicity Built-In FlashArray offers game-changing management simplicity that makes storage installation, configuration, provisioning and migration a snap. No more managing performance, RAID, tiers or caching. Achieve optimal application performance without any tuning at any layer. Manage the FlashArray the way you like it: Web-based GUI, CLI, VMware vCenter, Rest API, or OpenStack.

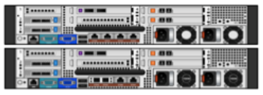
	FA-405	FA-420	FA-450
FRONT VIEW (dual controllers)			
REAR VIEW (dual controllers)			
CAPACITY	• Up to 40+ TBs effective capacity • 2.75-11 TBs raw capacity	• Up to 125+ TBs effective capacity • 11-35 TBs raw capacity	• Up to 250+ TBs effective capacity • 34-70 TBs raw capacity
	Effective capacity assumes HA, RAID, and metadata overhead, GB-to-GiB conversion, and includes benefit of data reduction with always-on inline deduplication, compression & pattern removal. Average data reduction is calculated at 6-to-1. Some customers see data reduction in excess of 20-to-1. Effective capacity has no upper limit and will vary depending on workload.		
PERFORMANCE	• Up to 100,000 **32K** IOPS @ <1ms average latency • Up to 3 GB/s bandwidth	• Up to 150,000 **32K** IOPS @ <1ms average latency • Up to 5 GB/s bandwidth	• Up to 200,000 **32K** IOPS @ <1ms average latency • Up to 7 GB/s bandwidth
	Why does Pure Storage quote 32K, not 4K IOPS? The industry commonly markets 4K IOPS benchmark to make numbers look high, but real-world environments are dominated by IO sizes of 32K or larger. Pure Storage has optimized the FlashArray for the real-world. FlashArray adapts automatically to 512B-32KB IO for superior performance, scalability, and data reduction.		
HOST CONNECTIVITY	• 8 Gb/s Fibre Channel • 10 Gb/s Ethernet iSCSI • Replication ports	• 8 Gb/s Fibre Channel • 10 Gb/s Ethernet iSCSI • Expansion slot (FC or iSCSI) • Replication ports	• 16 Gb/s Fibre Channel • 10 Gb/s Ethernet iSCSI • Expansion slot (FC or iSCSI) • Replication ports

Table 1. Pure Storage FlashArray 400 Series Specifications.

Start Small and Grow Online

FlashArray scales from smaller workloads to data center-wide consolidation. And because upgrading performance and capacity on the FlashArray is always non-disruptive, you can start small and grow without impacting mission-critical applications. Coupled with Forever Flash, a new business model for storage acquisition and lifecycles, FlashArray provides a simple and economical approach to evolutionary storage that extends the useful life of an array and does away with the incumbent storage vendor practices of forklift upgrades and maintenance extortion.

Windows PowerShell Version Verification

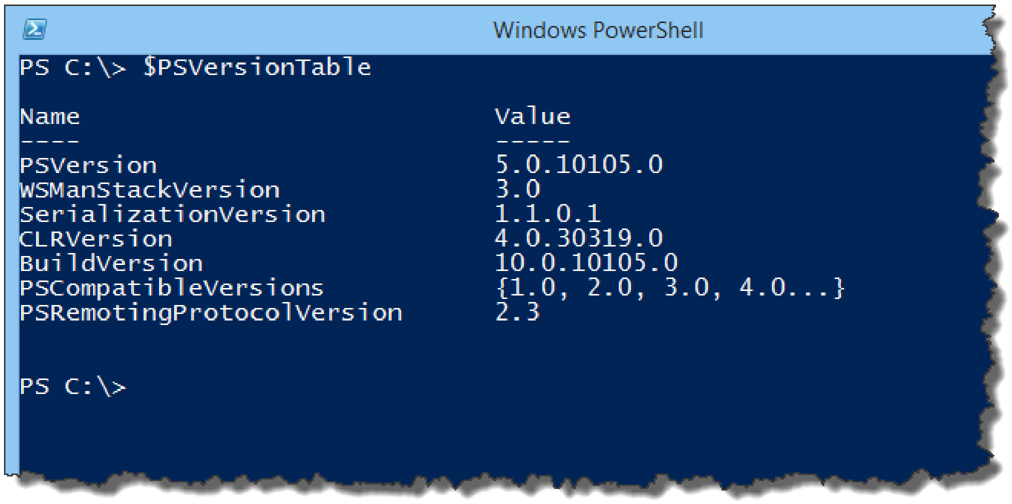
The SDK works with all of the versions of Windows that support PowerShell 3.0 and higher. The SDK can be used from Windows Client or Server editions. To determine what version of Windows and PowerShell you may be running we compiled the following table to provide some PowerShell version history, default Windows version and supported Windows versions below.

PowerShell Version	Release Date	Default Windows Version(s)	Supported On Windows Version(s)
1.0	November 2006	Windows Server 2008	Windows XP (SP2, SP3), Windows Server 2003 (SP1/SP2), Windows Server 2003 R2, Windows Vista (SP2),
2.0	October 2009	Windows 7, Windows Server 2008, Windows Server 2008 R2	Available in all newer Windows versions.
3.0 Windows Management Framework 3.0; backwardly compatible with PowerShell 2.0	September 2012	Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012	Windows XP (SP3), Windows Server 2003 (SP2), Windows Vista (SP1, SP2), Windows Server 2008 (SP1/SP2)
4.0 Windows Management Framework 4.0; backwardly compatible with PowerShell 2.0, 3.0	October 2013	Windows 8.1, Windows Server 2012 R2	Windows 7 (SP1), Windows Server 2008 R2 (SP1), Windows Server 2012
5.0 Windows Management Framework 5.0; backwardly compatible with PowerShell 2.0, 3.0, 4.0	Latest Preview April 2015	Windows 10	Windows 8.1, Windows Server 2016, Windows Server 2012 R2

Table 2. Windows PowerShell Versions.

To find out the version of Windows PowerShell installed in your environment open up a Windows PowerShell session and type `$PSVersionTable` this will show all details as seen in the screenshot below. The most important Name to check is `PSVersion`, the version below shows 5.0.10105.0 (PowerShell 5.0 – April 2015

Preview).


Windows PowerShell

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.0.10105.0
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.0
BuildVersion                   10.0.10105.0
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
PSRemotingProtocolVersion      2.3


PS C:\>
```

Figure 2. $PSVersionTable example.

The minimum version of Windows PowerShell required is 3.0 to use the Pure Storage PowerShell SDK. The SDK has also been tested with Windows PowerShell 4.0 and 5.0 (Preview).

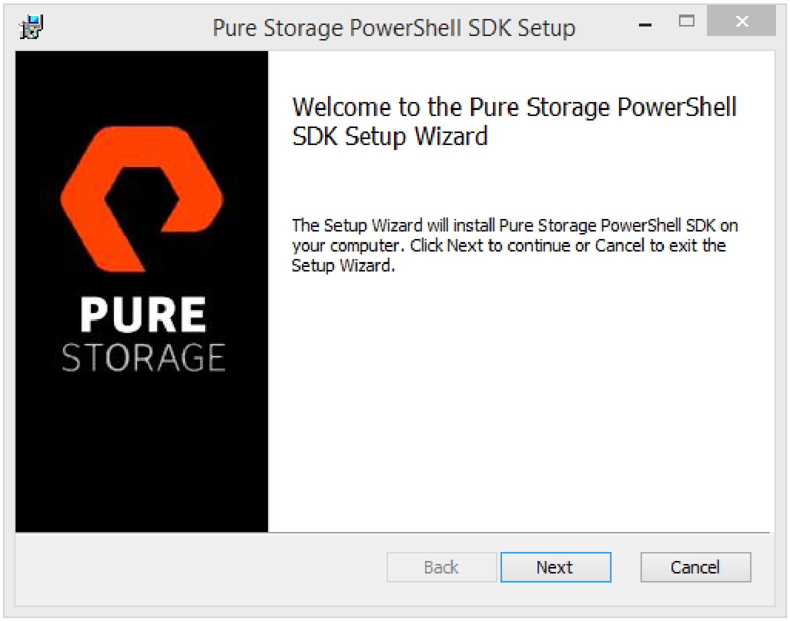
Setup Instructions

The first step before installing the new PowerShell SDK is to remove the Pure Storage PowerShell Toolkit. The latest version of the Toolkit is version 2.8.0.430, any version you may be running that is equal or below this should be removed before continuing. A new version of the Pure Storage PowerShell Toolkit 3.0 will be released that supports the use of the PowerShell SDK.

Download the PowerShell SDK from the Pure1 Community and once completed run the installer (PurePowerShellSDKInstaller.msi) as Administrator. When the setup begins you will walk through the following steps of the setup wizard.

Step 1 – Welcome Message

Click 'Next' to continue.



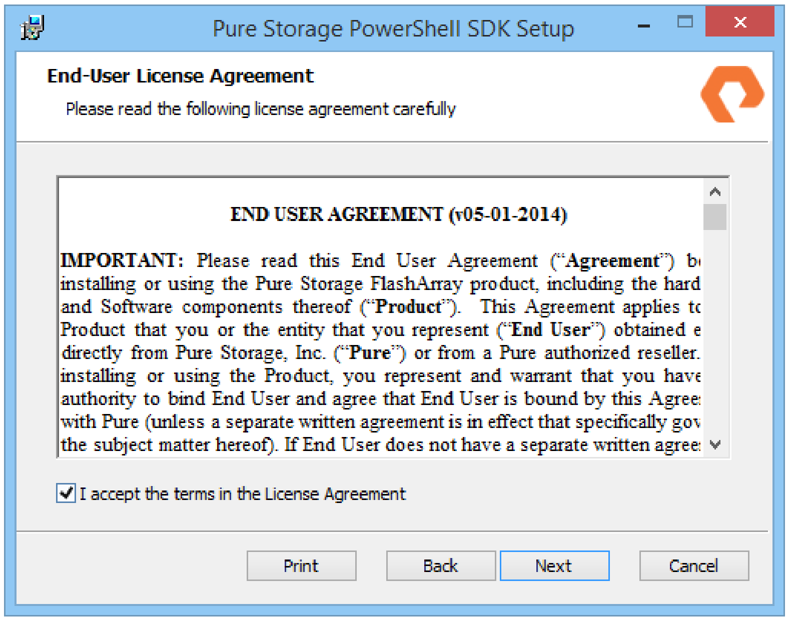
Welcome to the Pure Storage PowerShell SDK Setup Wizard

The Setup Wizard will install Pure Storage PowerShell SDK on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 3. Welcome step of the SDK Setup Wizard.

Step 2 – End-User License Agreement

Select "I accept the terms in the License Agreement" to continue.



End-User License Agreement

Please read the following license agreement carefully

END USER AGREEMENT (v05-01-2014)

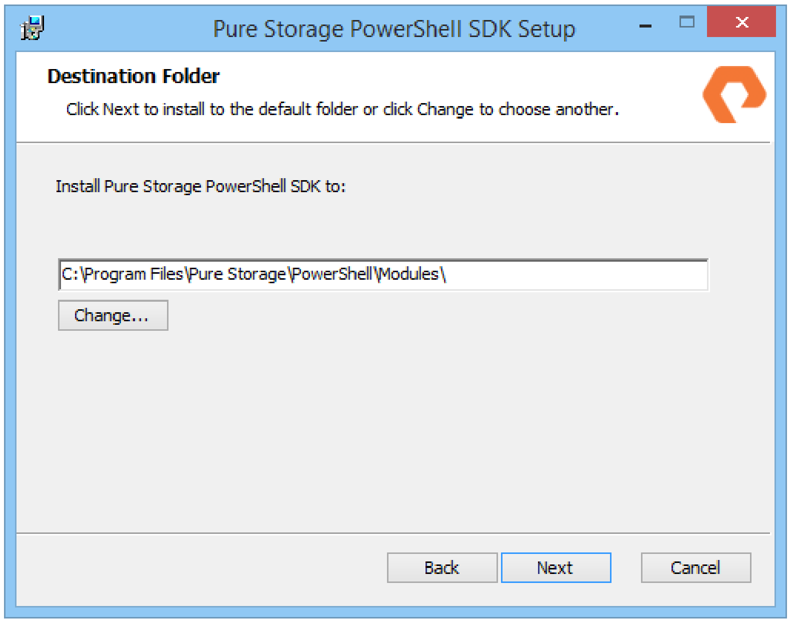
IMPORTANT: Please read this End User Agreement ("**Agreement**") b installing or using the Pure Storage FlashArray product, including the hard and Software components thereof ("**Product**"). This Agreement applies tc Product that you or the entity that you represent ("**End User**") obtained e directly from Pure Storage, Inc. ("**Pure**") or from a Pure authorized reseller. installing or using the Product, you represent and warrant that you have authority to bind End User and agree that End User is bound by this Agree with Pure (unless a separate written agreement is in effect that specifically gov the subject matter hereof). If End User does not have a separate written agree

☑ I accept the terms in the License Agreement

Figure 4. EULA step of the SDK Setup Wizard.

Step 3 – Destination Folder

You can change the location of the folder but we recommend accepting the default folder location.



Destination Folder

Click Next to install to the default folder or click Change to choose another.

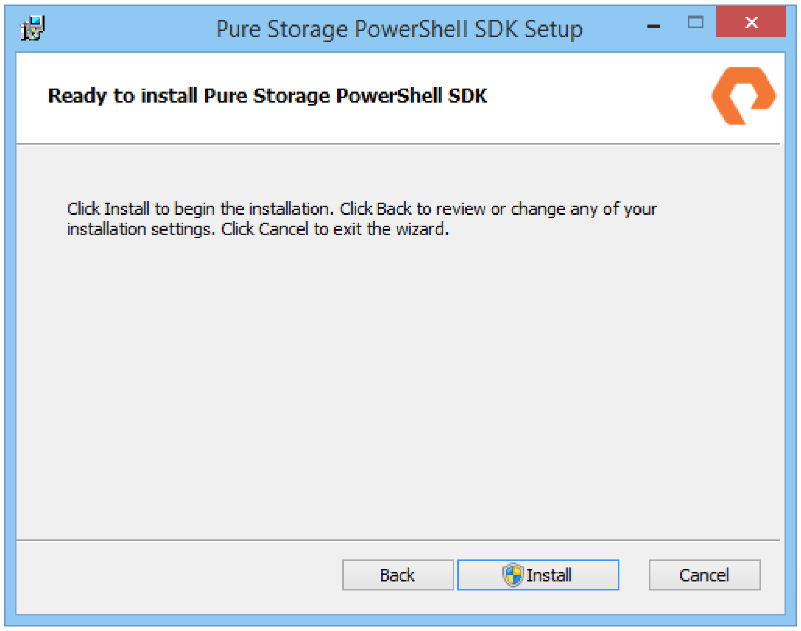
Install Pure Storage PowerShell SDK to:

C:\Program Files\Pure Storage\PowerShell\Modules\

Figure 5. Destination Folder step of the SDK Setup WIzard.

Step 4 – Ready to install Pure Storage PowerShell SDK

Click 'Install' to complete the setup.

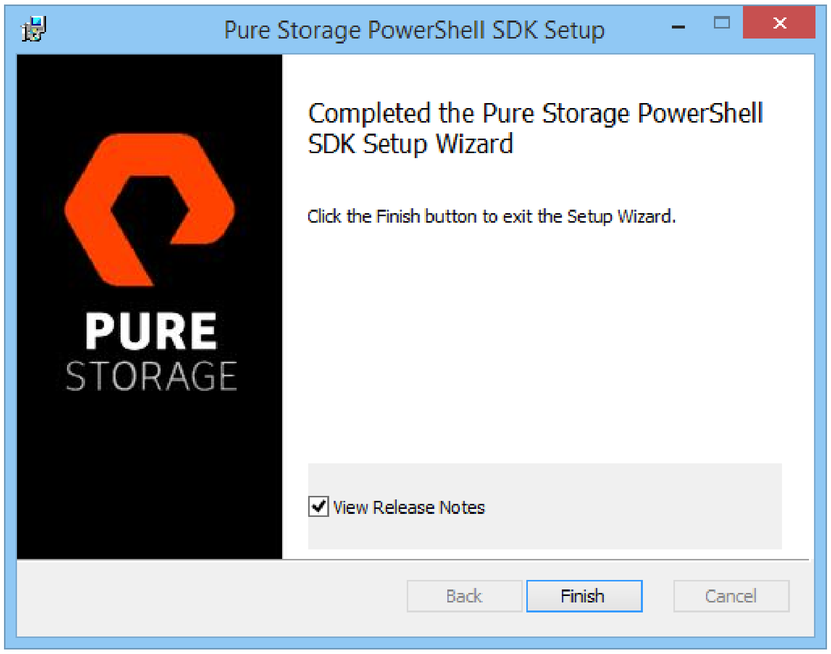


Ready to install Pure Storage PowerShell SDK

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Figure 6. Install step of the SDK Setup WIzard.

When the Pure Storage PowerShell SDK has completed setup you will see the following screen. We recommend that you select "View Release Notes", this will open up a release_notes.txt file using Notepad to display Release Compatibility, Installation/Uninstallation and any Known Issues.


Pure Storage PowerShell SDK Setup

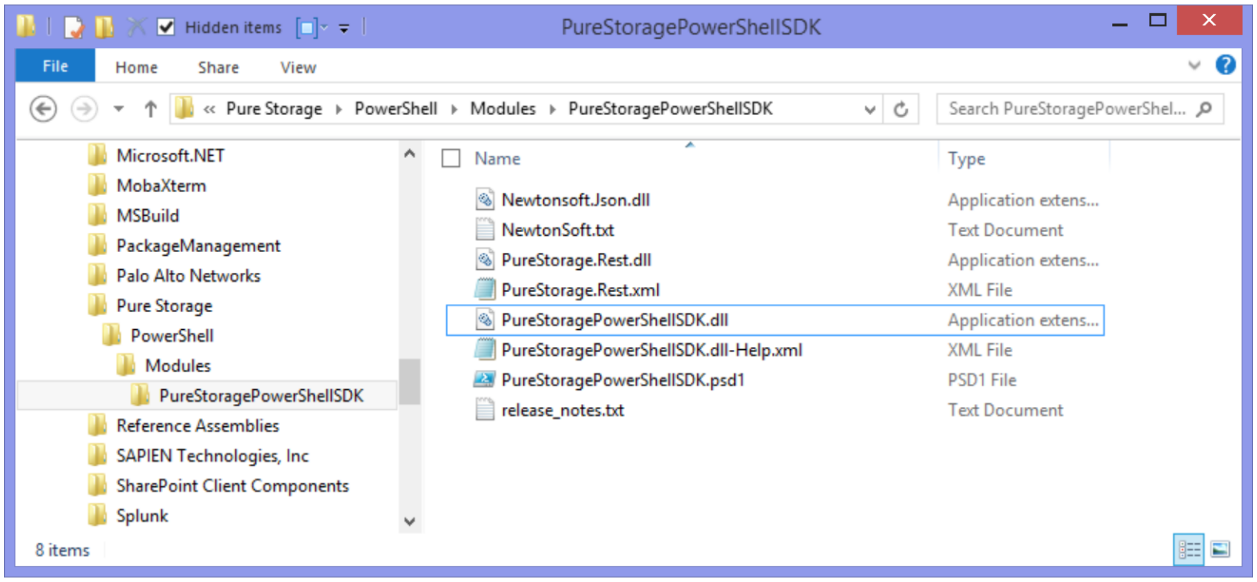
Completed the Pure Storage PowerShell SDK Setup Wizard

Click the Finish button to exit the Setup Wizard.

☑ View Release Notes

Figure 7. SDK Setup WIzard completed.

Below is the view of the default installation directory with all of the deployed files.



Name	Type
Newtonsoft.Json.dll	Application extens...
NewtonSoft.txt	Text Document
PureStorage.Rest.dll	Application extens...
PureStorage.Rest.xml	XML File
PureStoragePowerShellSDK.dll	Application extens...
PureStoragePowerShellSDK.dll-Help.xml	XML File
PureStoragePowerShellSDK.psd1	PSD1 File
release_notes.txt	Text Document

Figure 8. View of the installed binaries in the destination folder.

Installation of the Pure Storage PowerShell SDK 1.0 is now complete. Using the Windows Control Panel > Programs and Features the newly installed SDK can be seen.

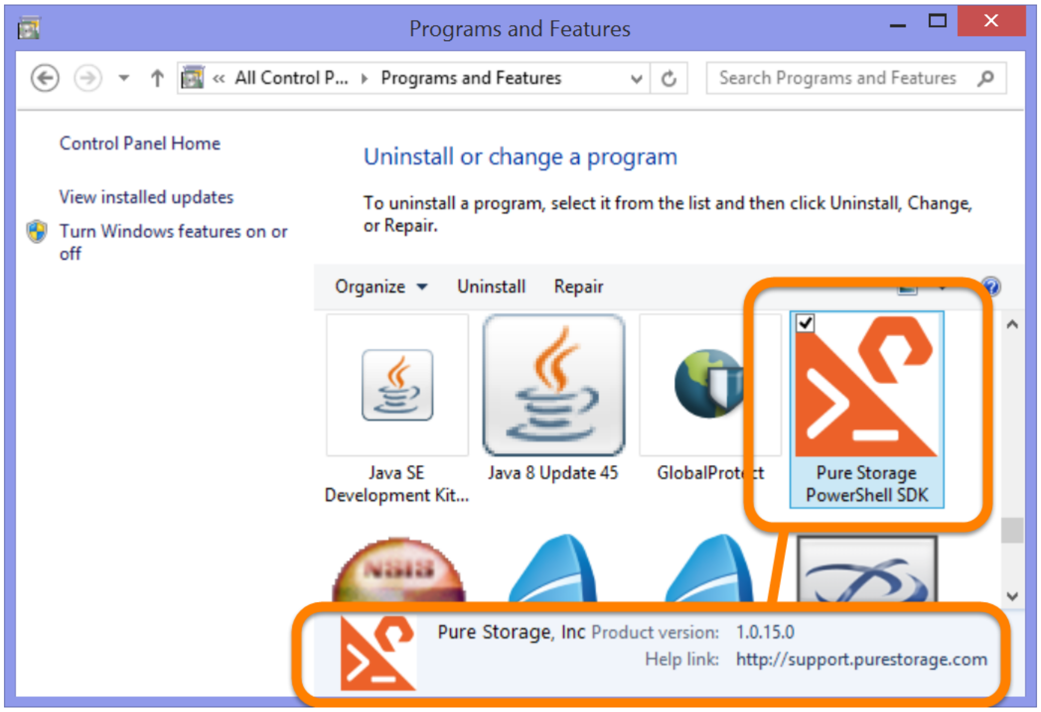

Programs and Features
All Control P... › Programs and Features
Search Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Uninstall or change a program
To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.
Organize
Uninstall
Repair
Java SE Development Kit...
Java 8 Update 45
GlobalProtect
Pure Storage PowerShell SDK
Pure Storage, Inc Product version: 1.0.15.0
Help link: http://support.purestorage.com

Figure 9. Windows Programs and Features.

If for any reason you wish to uninstall the Pure Storage PowerShell SDK use the Uninstall option provided by Programs and Features.

Now that we have successfully installed the SDK we can move on to using some of the basic PowerShell cmdlets.

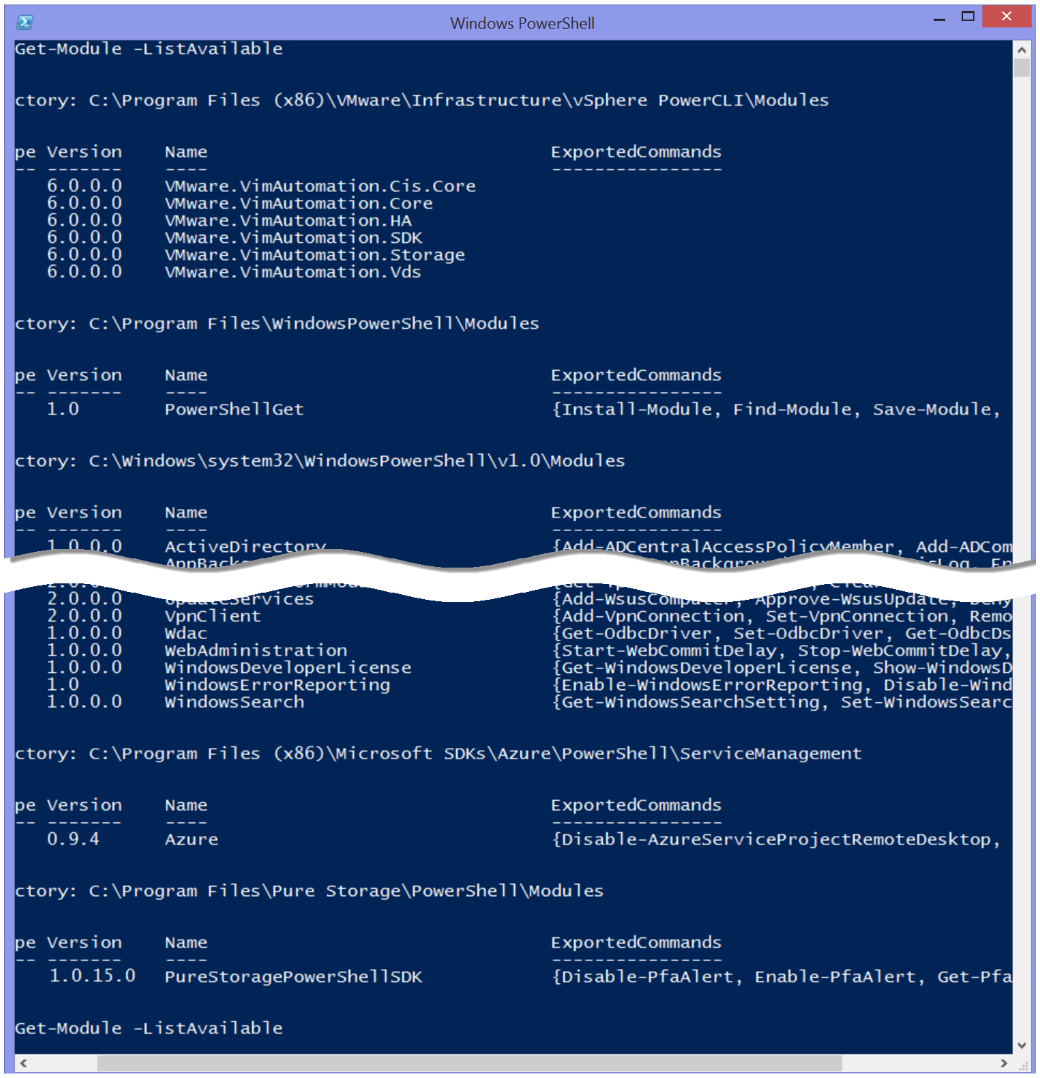
Getting Started

Start a Windows PowerShell session as Administrator. The reason to start the session as Administrator is so that help can be downloaded using Update-Help. The first task to perform is to run:

```
Get-Module –ListAvailable
```

Figure 10 shows the results for all available Modules and at the bottom is the PureStoragePowerShellSDK.

> **!** The screenshot in Figure 10 shows a listing provided by Windows PowerShell 5.0. Depending on the version of Windows PowerShell you are using the view may be slightly different.



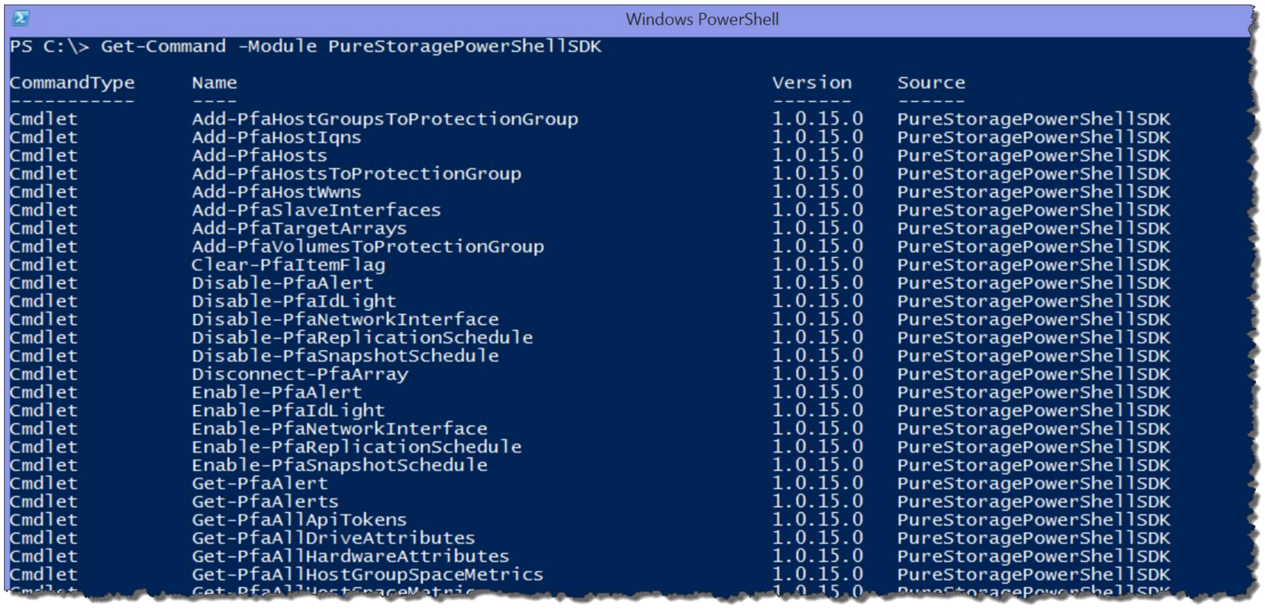
```
Get-Module -ListAvailable


ctory: C:\Program Files (x86)\VMware\Infrastructure\vSphere PowerCLI\Modules


pe Version    Name                                ExportedCommands
-- -------    ----                                ----------------
   6.0.0.0    VMware.VimAutomation.Cis.Core
   6.0.0.0    VMware.VimAutomation.Core
   6.0.0.0    VMware.VimAutomation.HA
   6.0.0.0    VMware.VimAutomation.SDK
   6.0.0.0    VMware.VimAutomation.Storage
   6.0.0.0    VMware.VimAutomation.Vds


ctory: C:\Program Files\WindowsPowerShell\Modules


pe Version    Name                                ExportedCommands
-- -------    ----                                ----------------
   1.0        PowerShellGet                       {Install-Module, Find-Module, Save-Module,


ctory: C:\Windows\system32\WindowsPowerShell\v1.0\Modules


pe Version    Name                                ExportedCommands
-- -------    ----                                ----------------
   1.0.0.0    ActiveDirectory                     {Add-ADCentralAccessPolicyMember, Add-ADCom
   2.0.0.0    UpdateServices                      {Add-WsusComputer, Approve-WsusUpdate, Deny
   2.0.0.0    VpnClient                           {Add-VpnConnection, Set-VpnConnection, Remo
   1.0.0.0    Wdac                                {Get-OdbcDriver, Set-OdbcDriver, Get-OdbcDs
   1.0.0.0    WebAdministration                   {Start-WebCommitDelay, Stop-WebCommitDelay,
   1.0.0.0    WindowsDeveloperLicense             {Get-WindowsDeveloperLicense, Show-WindowsD
   1.0        WindowsErrorReporting               {Enable-WindowsErrorReporting, Disable-Wind
   1.0.0.0    WindowsSearch                       {Get-WindowsSearchSetting, Set-WindowsSearc


ctory: C:\Program Files (x86)\Microsoft SDKs\Azure\PowerShell\ServiceManagement


pe Version    Name                                ExportedCommands
-- -------    ----                                ----------------
   0.9.4      Azure                               {Disable-AzureServiceProjectRemoteDesktop,


ctory: C:\Program Files\Pure Storage\PowerShell\Modules


pe Version    Name                                ExportedCommands
-- -------    ----                                ----------------
   1.0.15.0   PureStoragePowerShellSDK            {Disable-PfaAlert, Enable-PfaAlert, Get-Pfa


Get-Module -ListAvailable
```

Figure 10. Get-Module -ListAvailable.

Now run `$env:PSModulePath` which shows that the default folder location from the installation, C:\Program Files\Pure Storage\PowerShell\Modules is listed. During the installation the `PSModulePath` was updated to include the Pure Storage PowerShell folder so that using `Import-Module` is not necessary.

To test out that the Pure Storage PowerShell SDK cmdlets are visible enter the below command to see a list of the cmdets.

```
Get-Command -Module PureStoragePowerShellSDK
```

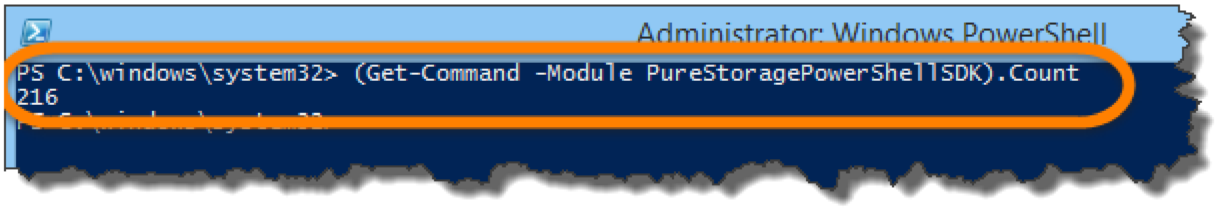


```
Windows PowerShell
PS C:\> Get-Command -Module PureStoragePowerShellSDK

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Add-PfaHostGroupsToProtectionGroup                 1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaHostIqns                                    1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaHosts                                       1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaHostsToProtectionGroup                      1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaHostWwns                                    1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaSlaveInterfaces                             1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaTargetArrays                                1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Add-PfaVolumesToProtectionGroup                    1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Clear-PfaItemFlag                                  1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Disable-PfaAlert                                   1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Disable-PfaIdLight                                 1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Disable-PfaNetworkInterface                        1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Disable-PfaReplicationSchedule                     1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Disable-PfaSnapshotSchedule                        1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Disconnect-PfaArray                                1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Enable-PfaAlert                                    1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Enable-PfaIdLight                                  1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Enable-PfaNetworkInterface                         1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Enable-PfaReplicationSchedule                      1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Enable-PfaSnapshotSchedule                         1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAlert                                       1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAlerts                                      1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllApiTokens                                1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllDriveAttributes                          1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllHardwareAttributes                       1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllHostGroupSpaceMetrics                    1.0.15.0   PureStoragePowerShellSDK
```

Figure 11. Get-Command.

You can also get the number of cmdlets in the SDK by modifying the Get-Command with the below. After running the command you will see that the SDK has 216 cmdlets.

```
(Get-Command -Module PureStoragePowerShellSDK).Count
```

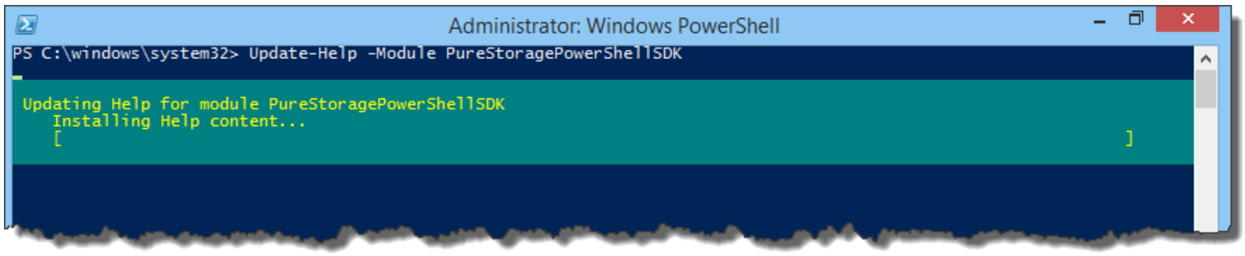


```
Administrator: Windows PowerShell
PS C:\windows\system32> (Get-Command -Module PureStoragePowerShellSDK).Count
216
```

Figure 12. Getting the count of cmdlets with Get-Command.

Before we start discussing the use of specific SDK cmdlets the help should be updated to the most recent version. To perform this task use:

```
Update-Help -Module PureStorageePowerShellSDK
```



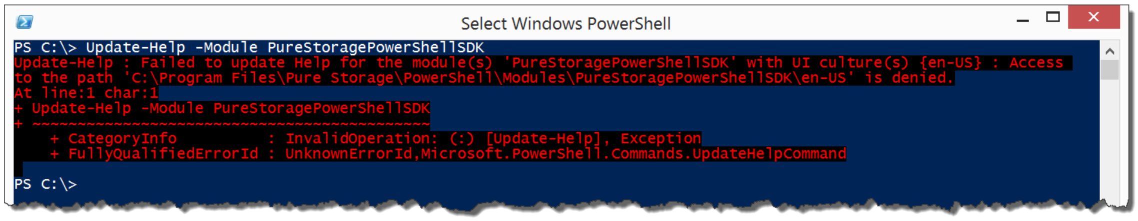
```
Administrator: Windows PowerShell
PS C:\windows\system32> Update-Help -Module PureStoragePowerShellSDK

 Updating Help for module PureStoragePowerShellSDK
    Installing Help content...
    [                                                                          ]
```

Figure 13. Update-Help.

`Update-Help` will connect to a URL that will download the latest Help files and add them to the installation folder. The reason running this cmdlet requires Windows PowerShell as Administrator is so that a new directory can be created. This operation happens very quickly. It is recommended that you run Update-Help periodically to be sure you are using the latest help information.

!

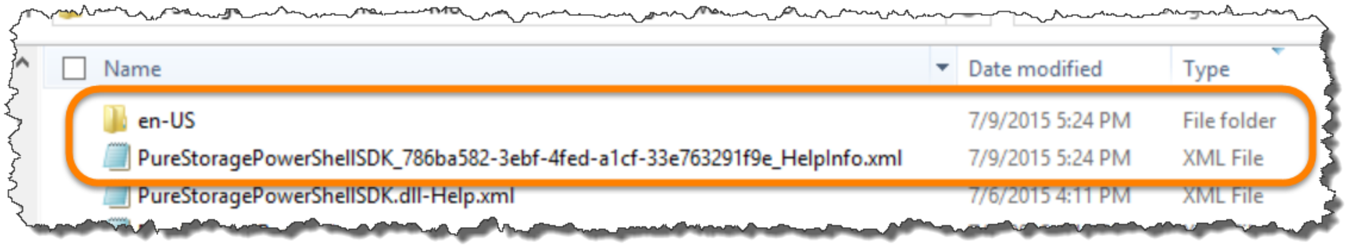
As mentioned earlier the only reason to start a Windows PowerShell session as Administrator is to be able to execute `Update-Help`. Figure 15 illustrates the error.



```
PS C:\> Update-Help -Module PureStoragePowerShellSDK
Update-Help : Failed to update Help for the module(s) 'PureStoragePowerShellSDK' with UI culture(s) {en-US} : Access
to the path 'C:\Program Files\Pure Storage\PowerShell\Modules\PureStoragePowerShellSDK\en-US' is denied.
At line:1 char:1
+ Update-Help -Module PureStoragePowerShellSDK
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [Update-Help], Exception
    + FullyQualifiedErrorId : UnknownErrorId,Microsoft.PowerShell.Commands.UpdateHelpCommand

PS C:\>
```

Figure 14. Update-Help Error.

To verify whether Update-Help downloaded and installed you can navigate to the installation directory check for the following folder and file shown in Figure 15.

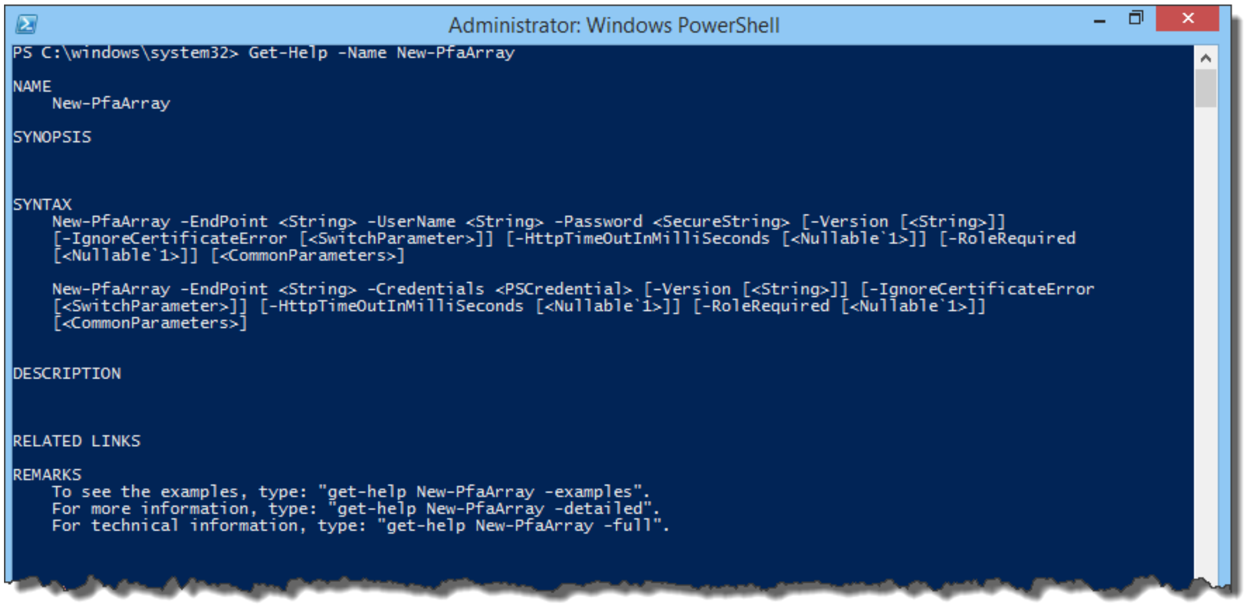


Name	Date modified	Type
en-US	7/9/2015 5:24 PM	File folder
PureStoragePowerShellSDK_786ba582-3ebf-4fed-a1cf-33e763291f9e_HelpInfo.xml	7/9/2015 5:24 PM	XML File
PureStoragePowerShellSDK.dll-Help.xml	7/6/2015 4:11 PM	XML File

Figure 15. Verify Update-Help.

Figure 16 illustrates how to use the Get-Help cmdlet to find specific assistance with an SDK cmdlet. By adding `-Full`, `-Detailed` or `-Examples` more helpful information and examples will be displayed.

```
Get-Help New-PfaArray
```



```
PS C:\windows\system32> Get-Help -Name New-PfaArray

NAME
    New-PfaArray

SYNOPSIS


SYNTAX
    New-PfaArray -EndPoint <String> -UserName <String> -Password <SecureString> [-Version [<String>]]
    [-IgnoreCertificateError [<SwitchParameter>]] [-HttpTimeOutInMilliSeconds [<Nullable`1>]] [-RoleRequired
    [<Nullable`1>]] [<CommonParameters>]

    New-PfaArray -EndPoint <String> -Credentials <PSCredential> [-Version [<String>]] [-IgnoreCertificateError
    [<SwitchParameter>]] [-HttpTimeOutInMilliSeconds [<Nullable`1>]] [-RoleRequired [<Nullable`1>]]
    [<CommonParameters>]


DESCRIPTION


RELATED LINKS

REMARKS
    To see the examples, type: "get-help New-PfaArray -examples".
    For more information, type: "get-help New-PfaArray -detailed".
    For technical information, type: "get-help New-PfaArray -full".
```

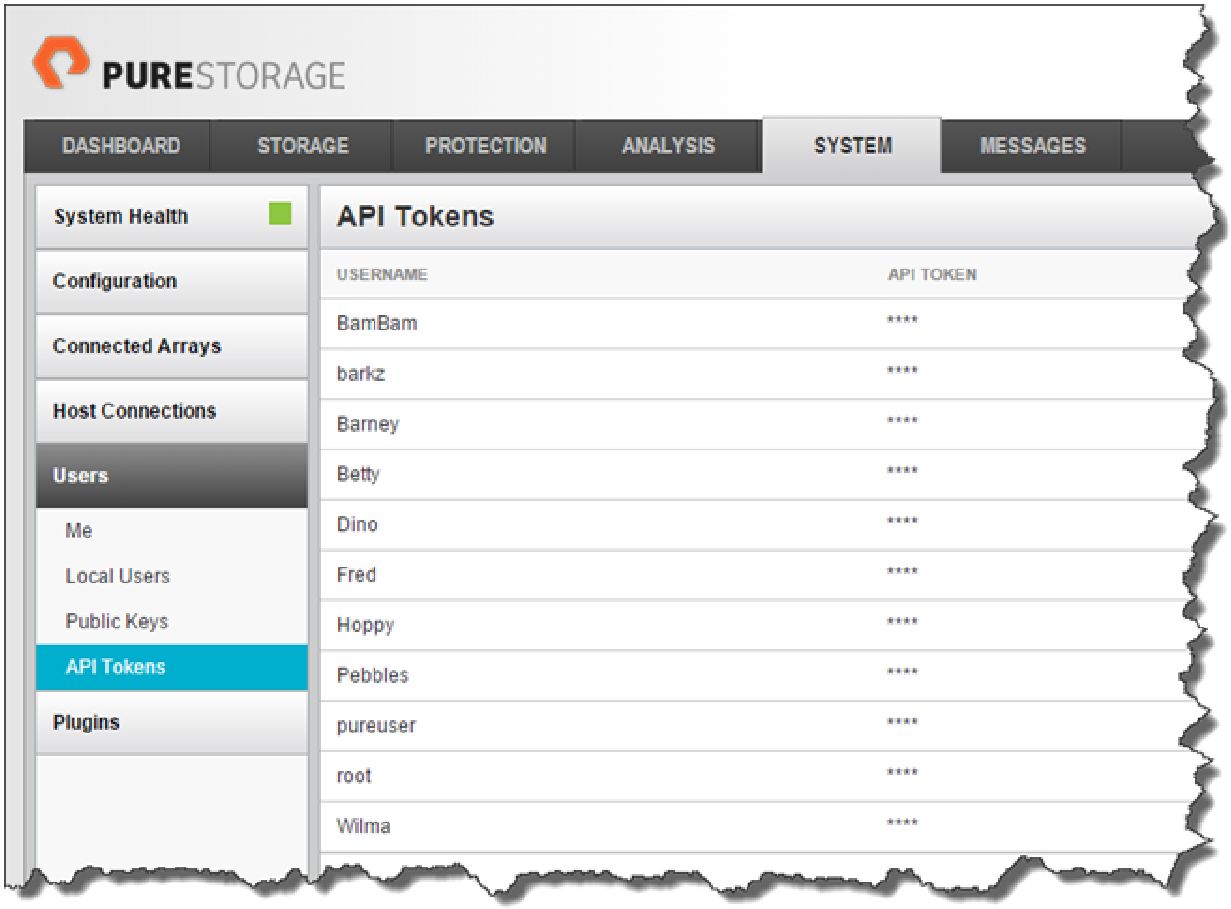
Figure 16. Get-Help.

Any of the cmdlets that are part of the Pure Storage PowerShell SDK are all preceded with the `Pfa` acronym. This acronym signifies Pure Storage FlashArray so when reading any scripts from colleagues or samples you will know immediately that a particular cmdlet is interacting with a FlashArray.

Credentials

The Pure Storage PowerShell SDK uses the REST API which is based on an API Token authentication model. Individual users whether they are added to the FlashArray manually or leverages our Directory Services integration require an associated API Token in order to issue commands.

Figure 17 shows an example from one of our lab FlashArrays with a mixture of usernames from Active Directory (AD) and other default FlashArray accounts (Eg. pureuser). A connection is created to the FlashArray with credentials passed using either a PSCredential object or using `-Username` and `-Password` parameters of the `New-PfaArray` cmdlet. Part of the `New-PfaArray` cmdlet process is to verify the account has an API Token, can authenticate and then create the FlashArray object to be used in subsequent operations.



API Tokens

USERNAME	API TOKEN
BamBam	****
barkz	****
Barney	****
Betty	****
Dino	****
Fred	****
Hoppy	****
Pebbles	****
pureuser	****
root	****
Wilma	****

Figure 17. API Tokens.

When using the Directory Services integration, users need to be in one of the three directory groups, Read Only, Storage Admin or Array Admin. As long as a user exists in one of the groups the cmdlet will authenticate the user with the domain credentials and check for an API Token. If an API Token does not exist, one will be created automatically which will then be visible from the System > Users > API Token section of the Pure Storage Web Management tool, as shown in Figure 17.

When connecting to the Pure Storage FlashArray with a directory service account the format for entering the credentials is USERNAME and PASSWORD. It is not necessary to use the common format of DOMAIN\USERNAME. Since the FlashArray has been configured with the specific directory it will automatically use the designated domain name.

Connecting to the FlashArray

In the following section we will start performing the basic operations to set credentials for a specific user and creating a FlashArray object for use with subsequent PowerShell cmdlets. The tasks to complete will be the following:

1. Set credentials
2. Connect to the FlashArray
3. Get the FlashArray controllers

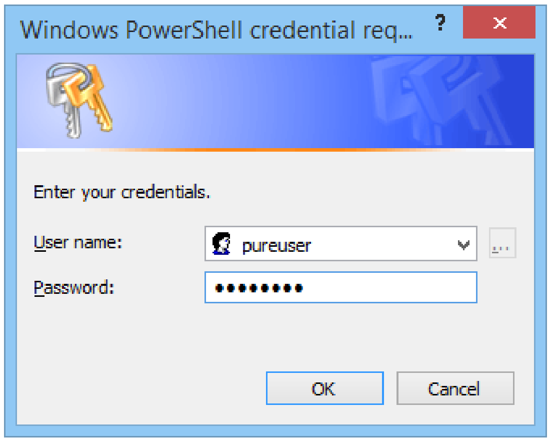
Here are the SDK cmdlets to use for each of the above steps.

Step 1

This line of script assigns a PSCredential to the $Creds variable so it can be used in subsequent cmdlets.

```
$Creds = Get-Credential
```

After executing this command a dialog box will pop-up to enter the credentials to use for the connection. Use the default FlashArray user pureuser with the password pureuser (or whatever update you may have made to the password).


Windows PowerShell credential req...

Enter your credentials.

User name: pureuser

Password: ••••••••

Figure 18. Credential dialog.

Step 2

This line of script assigns a new PfaArray object to the $FlashArray variable. The new object is created using the EndPoint parameter, this parameter can be either an IP Address or a resolved DNS name. Next we pass the $Creds variable that we assigned in the previous steps with the appropriate user to create a REST session with the FlashArray.

!

The example below contains a tick mark (`) which signifies a line continuation character in Windows PowerShell. Some of the examples in this document use the tick mark to ensure if you cut-n-paste from the document into a Windows PowerShell session that the line of script will work properly.

```
$FlashArray = New-PfaArray -EndPoint csg-fa420-2 -Credentials $Creds `
-IgnoreCertificateError
```

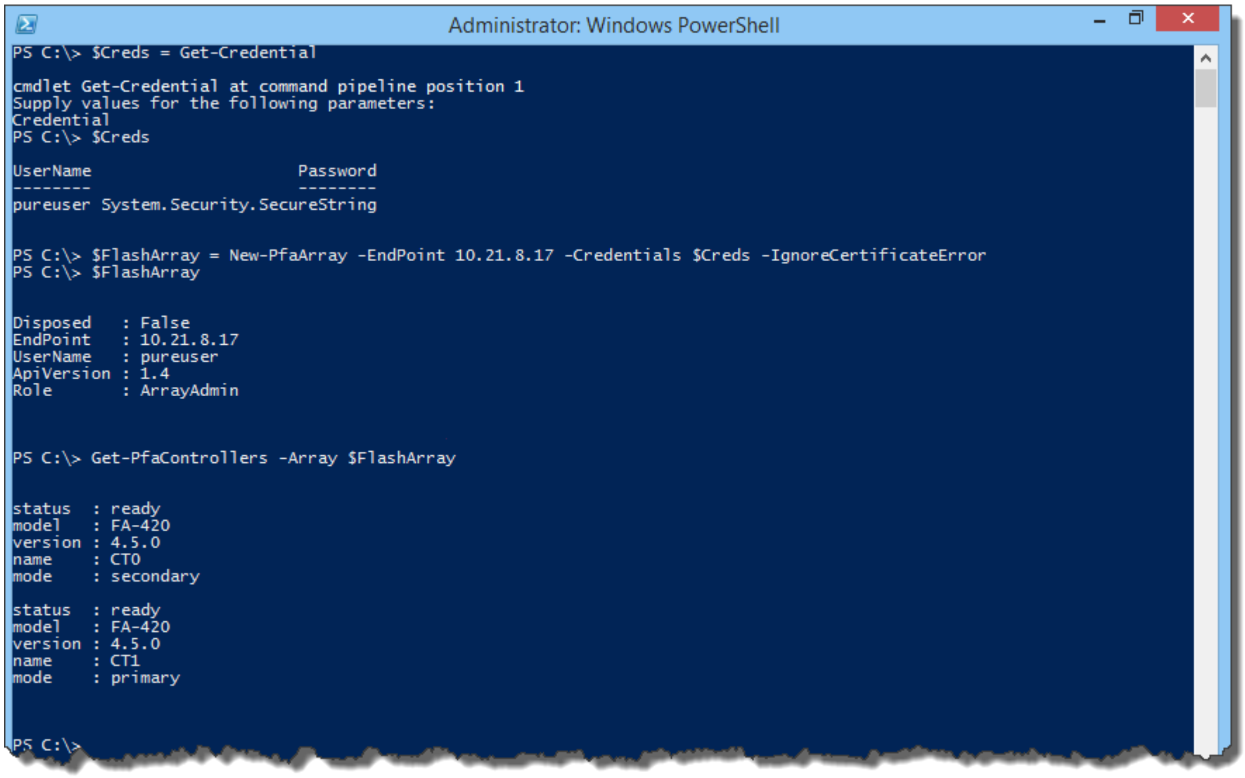
There are additional parameters for the `New-PfaArray` cmdlet that you can check using `Get-Help`.

The `IgnoreCertificateError` parameter is used here as the FlashArray being used does not have an SSL certificate setup. If you have an SSL certificate setup with the FlashArray you would not be required to use this parameter.

Step 3

This line of script gets the information for each of the controllers for a particular FlashArray. The `$FlashArray` variable is passed with the `–Array` parameter.

```
Get-PfaControllers –Array $FlashArray
```

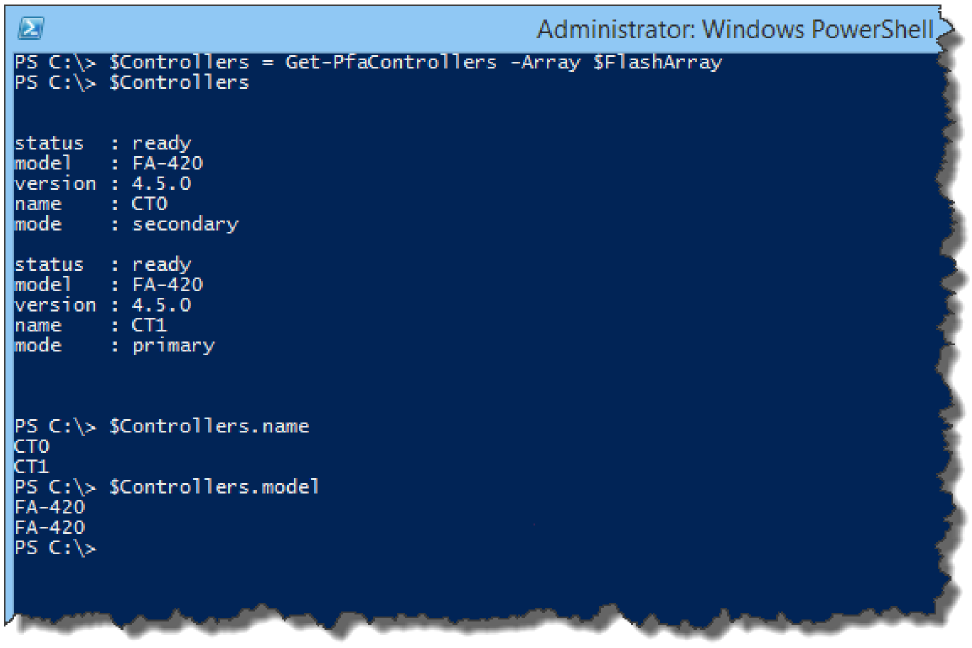


```
PS C:\> $Creds = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
PS C:\> $Creds

UserName                     Password
--------                     --------
pureuser System.Security.SecureString


PS C:\> $FlashArray = New-PfaArray -EndPoint 10.21.8.17 -Credentials $Creds -IgnoreCertificateError
PS C:\> $FlashArray


Disposed   : False
EndPoint   : 10.21.8.17
UserName   : pureuser
ApiVersion : 1.4
Role       : ArrayAdmin



PS C:\> Get-PfaControllers -Array $FlashArray


status  : ready
model   : FA-420
version : 4.5.0
name    : CT0
mode    : secondary

status  : ready
model   : FA-420
version : 4.5.0
name    : CT1
mode    : primary


PS C:\>
```

Figure 19. Sample of using Credentials, Connecting and retrieving controller information.

Any time information is retrieved (Get operations) from the FlashArray they are returned as PSObjects (PowerShell Objects) so that further formatting or other tasks can be performed. Below is an example of assigning the output from the `Get-PfaControllers` cmdlet to a `$Controllers` variable and then querying for the individual values.

```
$Controllers = Get-PfaControllers –Array $FlashArray
$Controllers
```


Administrator: Windows PowerShell

```
PS C:\> $Controllers = Get-PfaControllers -Array $FlashArray
PS C:\> $Controllers


status  : ready
model   : FA-420
version : 4.5.0
name    : CT0
mode    : secondary

status  : ready
model   : FA-420
version : 4.5.0
name    : CT1
mode    : primary


PS C:\> $Controllers.name
CT0
CT1
PS C:\> $Controllers.model
FA-420
FA-420
PS C:\>
```

Figure 20. Get-PfaControllers.

Working with Volumes

Now that we can create a connection to the FlashArray using New-PfaArray we can start adding commands. The next task to complete will be the following:

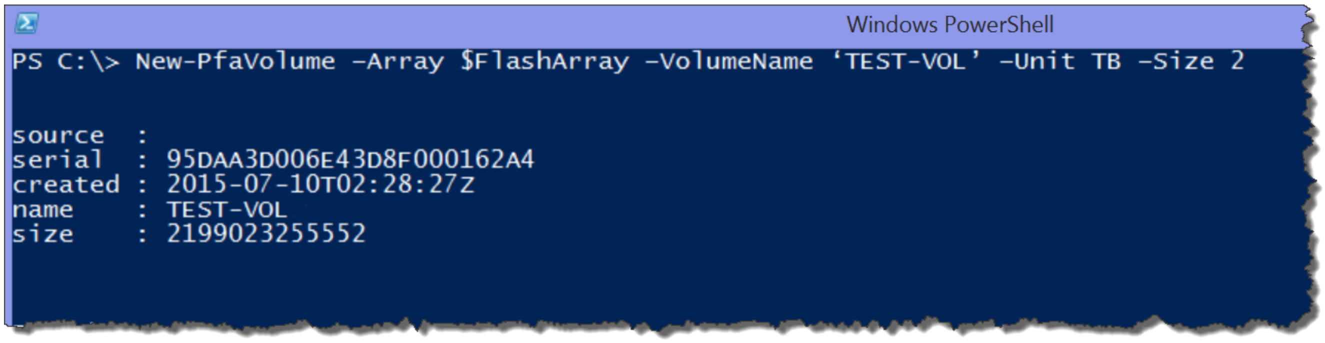
1. Create a new volume
2. Create several new volumes dynamically
3. Get the details of the new volume

Here are the SDK cmdlets to use for each of the above steps:

Step 1

This line of script will create a new 2TB volume named TEST-VOL. The `–Array` parameter takes the same `$FlashArray` object that was used in the previous sample.

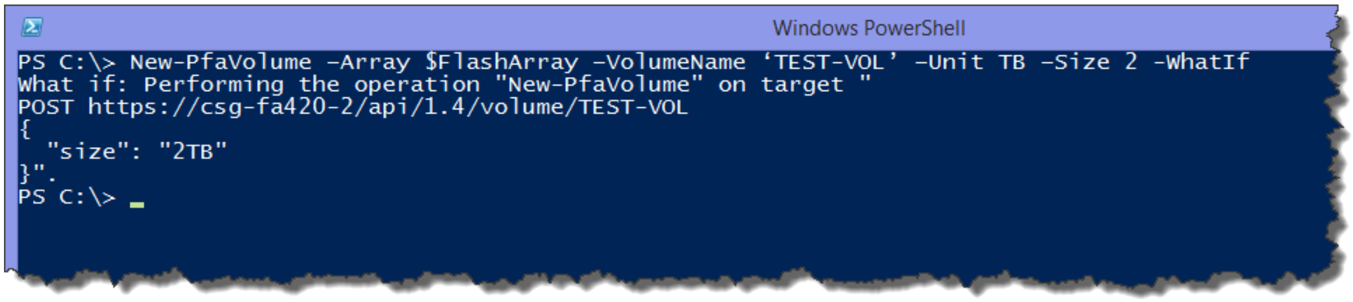
```
New-PfaVolume –Array $FlashArray –VolumeName 'TEST-VOL' –Unit TB –Size 2
```



```
PS C:\> New-PfaVolume -Array $FlashArray -VolumeName 'TEST-VOL' -Unit TB -Size 2


source  :
serial  : 95DAA3D006E43D8F000162A4
created : 2015-07-10T02:28:27Z
name    : TEST-VOL
size    : 2199023255552
```

Figure 21. New-PfaVolume example.

Each of the SDK cmdlets supports the `-WhatIf` parameter that shows the exact REST API call. See the example below using the same call with the `WhatIf` parameter



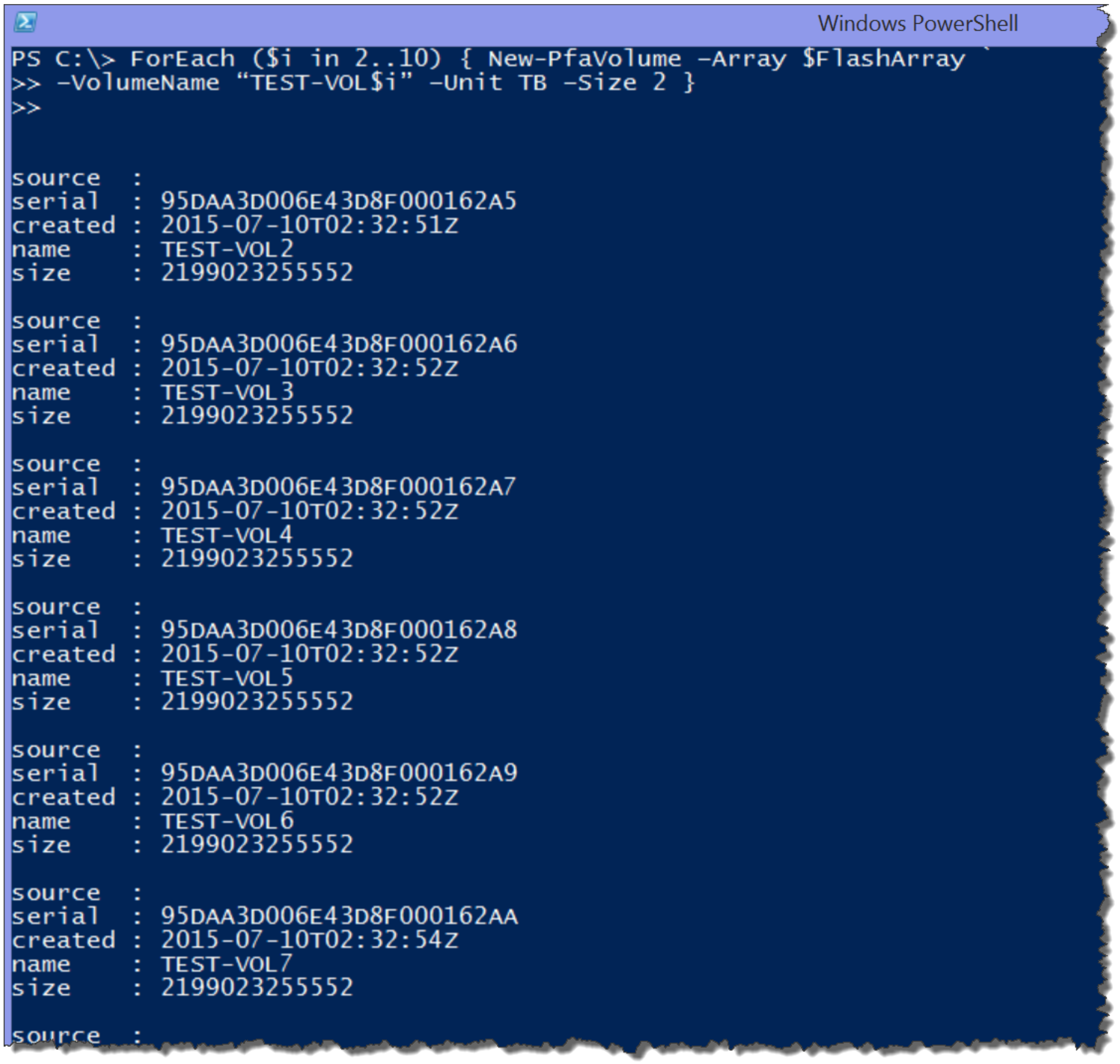
```
PS C:\> New-PfaVolume -Array $FlashArray -VolumeName 'TEST-VOL' -Unit TB -Size 2 -WhatIf
What if: Performing the operation "New-PfaVolume" on target "
POST https://csg-fa420-2/api/1.4/volume/TEST-VOL
{
  "size": "2TB"
}".
PS C:\> _
```

Figure 22. New-PfaVolume using -WhatIf parameter.

Step 2

In addition to create a single volume using the SDK, you can also create any number of volumes with a small amount of scripting. For example we will create 9 additional volumes with a single line of script.

```
ForEach ($i in 2..10) { New-PfaVolume -Array $FlashArray `
-VolumeName "TEST-VOL$i" -Unit TB -Size 2 }
```



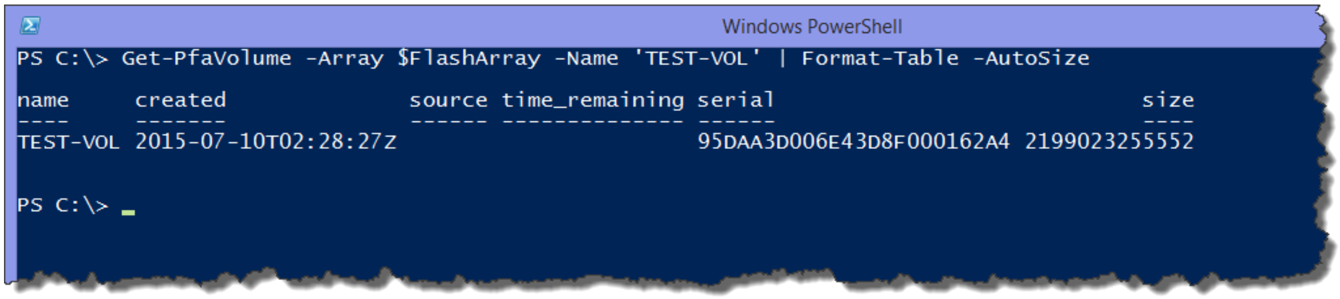
```
PS C:\> ForEach ($i in 2..10) { New-PfaVolume -Array $FlashArray `
>> -VolumeName "TEST-VOL$i" -Unit TB -Size 2 }
>>


source  :
serial  : 95DAA3D006E43D8F000162A5
created : 2015-07-10T02:32:51Z
name    : TEST-VOL2
size    : 2199023255552

source  :
serial  : 95DAA3D006E43D8F000162A6
created : 2015-07-10T02:32:52Z
name    : TEST-VOL3
size    : 2199023255552

source  :
serial  : 95DAA3D006E43D8F000162A7
created : 2015-07-10T02:32:52Z
name    : TEST-VOL4
size    : 2199023255552

source  :
serial  : 95DAA3D006E43D8F000162A8
created : 2015-07-10T02:32:52Z
name    : TEST-VOL5
size    : 2199023255552

source  :
serial  : 95DAA3D006E43D8F000162A9
created : 2015-07-10T02:32:52Z
name    : TEST-VOL6
size    : 2199023255552

source  :
serial  : 95DAA3D006E43D8F000162AA
created : 2015-07-10T02:32:54Z
name    : TEST-VOL7
size    : 2199023255552

source  :
```

Figure 23. Creating 9 additional volumes dynamically.

Step 3

This line of script is retrieving the details about the new volume. At the end of the first cmdlet we are using a pipeline (|) to take the retrieved data and pass it to the Format-Table cmdlet to customize the visual representation.

```
Get-PfaVolume -Array $FlashArray -Name 'TEST-VOL' | Format-Table `
-Autosize
```

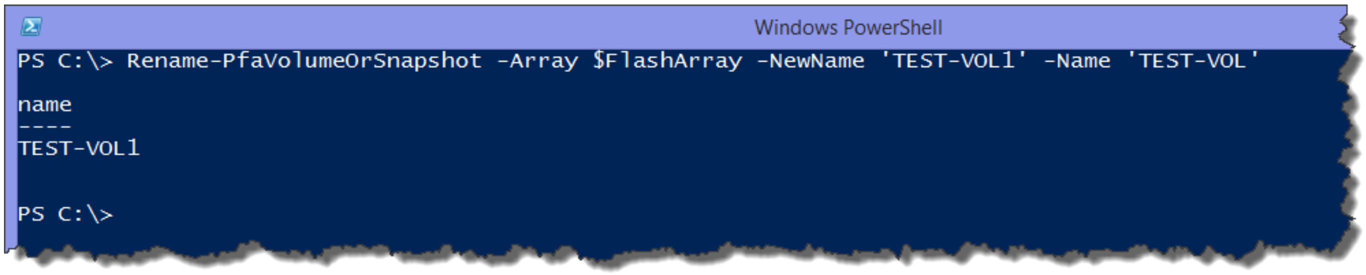


```
PS C:\> Get-PfaVolume -Array $FlashArray -Name 'TEST-VOL' | Format-Table -AutoSize

name     created              source time_remaining serial                   size
----     -------              ------ -------------- ------                   ----
TEST-VOL 2015-07-10T02:28:27Z                       95DAA3D006E43D8F000162A4 2199023255552


PS C:\> _
```

Figure 24. Get-PfaVolume example.

The script in Step 2 above used a ForEach loop to increment the variable $i to create TEST2, TEST3, TEST4, etc. To stay with the name convention for the volumes just created we can use the `Rename-PfaVolumeOrSnapshot` cmdlet to rename TEST to TEST1, so we will have volumes named TEST1 – TEST10.

```
Rename-PfaVolumeOrSnapshot –Array $FlashArray –NewName 'TEST-VOL1' `
–Name 'TEST-VOL'
```

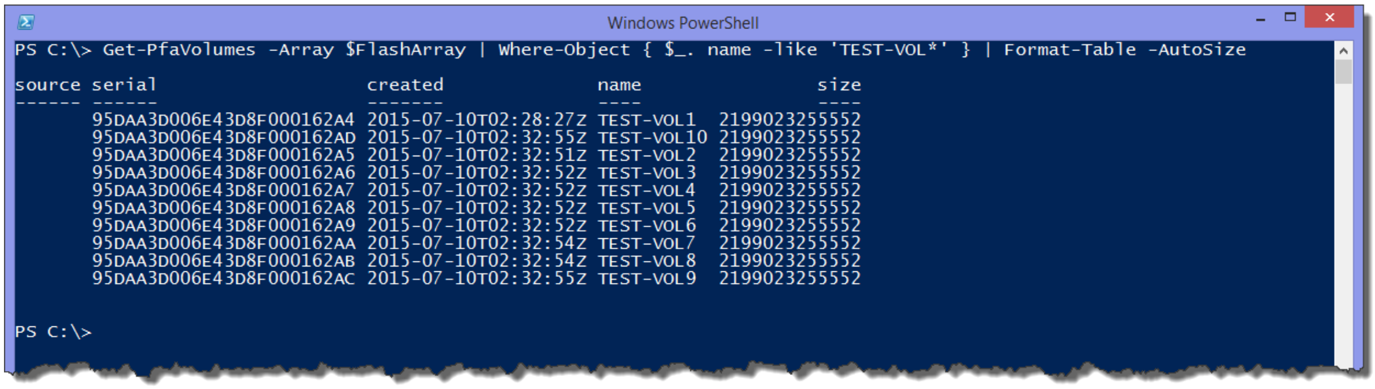


```
PS C:\> Rename-PfaVolumeOrSnapshot -Array $FlashArray -NewName 'TEST-VOL1' -Name 'TEST-VOL'

name
----
TEST-VOL1


PS C:\>
```

Figure 25. Rename-PfaVolumeOrSnapshot example.

Using the `Get-PfaVolumes` cmdlet we see all of the volumes named properly.

```
Get-PfaVolumes -Array $FlashArray | `
Where-Object { $_. name -like 'TEST-VOL*' } | Format-Table -AutoSize
```



```
PS C:\> Get-PfaVolumes -Array $FlashArray | Where-Object { $_. name -like 'TEST-VOL*' } | Format-Table -AutoSize

source serial                   created              name      size
------ ------                   -------              ----      ----
       95DAA3D006E43D8F000162A4 2015-07-10T02:28:27Z TEST-VOL1 2199023255552
       95DAA3D006E43D8F000162AD 2015-07-10T02:32:55Z TEST-VOL10 2199023255552
       95DAA3D006E43D8F000162A5 2015-07-10T02:32:51Z TEST-VOL2 2199023255552
       95DAA3D006E43D8F000162A6 2015-07-10T02:32:52Z TEST-VOL3 2199023255552
       95DAA3D006E43D8F000162A7 2015-07-10T02:32:52Z TEST-VOL4 2199023255552
       95DAA3D006E43D8F000162A8 2015-07-10T02:32:52Z TEST-VOL5 2199023255552
       95DAA3D006E43D8F000162A9 2015-07-10T02:32:52Z TEST-VOL6 2199023255552
       95DAA3D006E43D8F000162AA 2015-07-10T02:32:54Z TEST-VOL7 2199023255552
       95DAA3D006E43D8F000162AB 2015-07-10T02:32:54Z TEST-VOL8 2199023255552
       95DAA3D006E43D8F000162AC 2015-07-10T02:32:55Z TEST-VOL9 2199023255552


PS C:\>
```

Figure 26. Get-PfaVolumes example.

Before moving on, let's recap what we have learned so far:

- Assigned credentials to a variable using `Get-Credential`
- Created a persistent connection to the FlashArray using `New-PfaArray`
- Retrieved FlashArray controller information using `Get-PfaControllers`
- Created a new volume using `New-PfaVolume`
- Queried and formatted the volume details using `Get-PfaVolumes` and `Format-Table`
- Renamed existing volumes using `Rename-Pfavolume`

The next several sections will walk through connecting volumes to hosts and host group, creating snapshots, setting up protection groups and lastly monitoring the metrics.

Volume and Host Connections

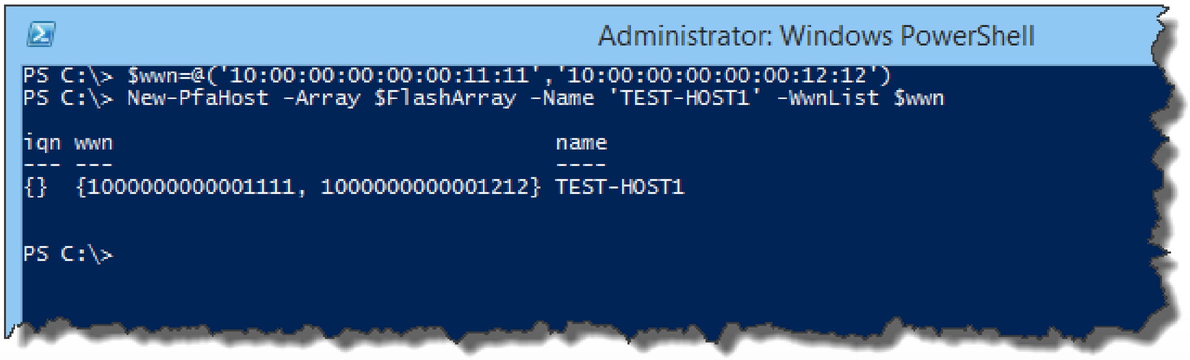
Previously we created 10 x 2TB volumes named TEST-VOL1 – TEST-VOL10. The next step is to create a Host or Host Group to connect the volumes for use. The next tasks to complete will be the following:

1. Create a new Host
2. Create a new Host Group and add the Host to the Host Group
3. Connect volumes to the Host and Host Group

Here are the SDK cmdlets to use for each of the above steps:

Step 1

```
$wwn=@('10:00:00:00:00:00:11:11','10:00:00:00:00:00:12:12')
New-PfaHost –Array $FlashArray –Name 'TEST-HOST1' –WwnList $wwn
```



```
PS C:\> $wwn=@('10:00:00:00:00:00:11:11','10:00:00:00:00:00:12:12')
PS C:\> New-PfaHost -Array $FlashArray -Name 'TEST-HOST1' -WwnList $wwn

iqn wwn                                  name
--- ---                                  ----
{}  {1000000000001111, 1000000000001212} TEST-HOST1


PS C:\>
```

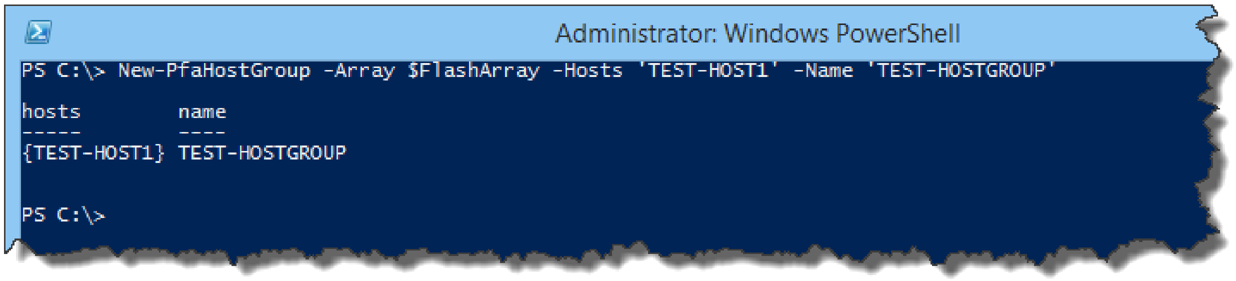
Figure 27. New-PfaHost example.

In the above example, a WWN array is set first to some sample WWNs. For script testing purposes it is ok to use self-created WWNs. The next cmdlet to run is the `New-PfaHost` with the required parameters. Just as with the `New-PfaVolume` cmdlet it is possible to script this so that a number of hosts are automatically created.

Step 2

The next step is to create a Host Group which we can add the new TEST-HOST1 created in Step 1. The use of a Host Group allows the creation of a virtual cluster that contains one or many hosts. Volumes can be connected to the Host Group which allows visibility across all hosts or a volume can be connected to a single host for access only by that host.

```
New-PfaHostGroup -Array $FlashArray -Hosts 'TEST-HOST1' `
-Name 'TEST-HOSTGROUP'
```



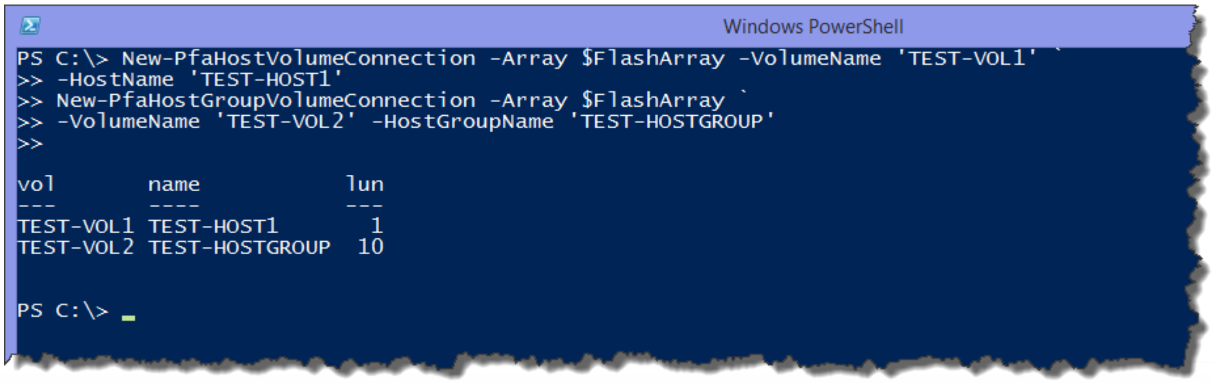
```
PS C:\> New-PfaHostGroup -Array $FlashArray -Hosts 'TEST-HOST1' -Name 'TEST-HOSTGROUP'

hosts        name
-----        ----
{TEST-HOST1} TEST-HOSTGROUP


PS C:\>
```

Figure 28. New-PfaHostGroup example.

Step 3

The final step is to connect a volume to the new host and a volume to the host group.

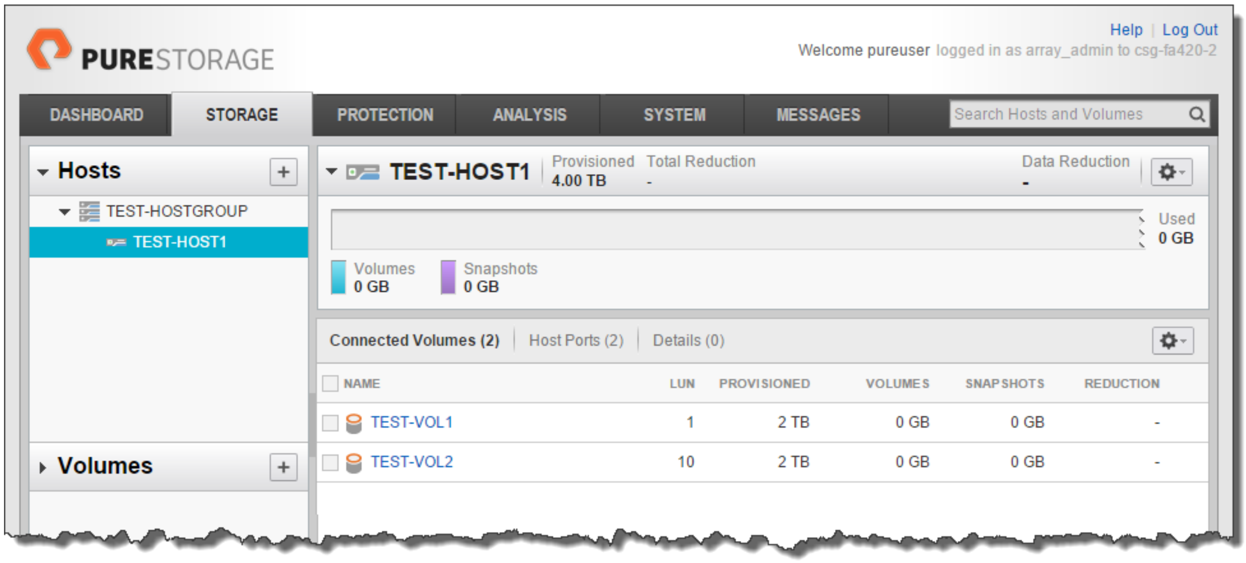
```
New-PfaHostVolumeConnection -Array $FlashArray -VolumeName 'TEST-VOL1' `
-HostName 'TEST-HOST1'
New-PfaHostGroupVolumeConnection -Array $FlashArray `
-VolumeName 'TEST-VOL2' -HostGroupName 'TEST-HOSTGROUP'
```



```
PS C:\> New-PfaHostVolumeConnection -Array $FlashArray -VolumeName 'TEST-VOL1' `
>> -HostName 'TEST-HOST1'
>> New-PfaHostGroupVolumeConnection -Array $FlashArray `
>> -VolumeName 'TEST-VOL2' -HostGroupName 'TEST-HOSTGROUP'
>>

vol       name           lun
---       ----           ---
TEST-VOL1 TEST-HOST1       1
TEST-VOL2 TEST-HOSTGROUP  10


PS C:\> _
```

Figure 29. New-PfaHostVolumeConnection and New-PfaHostGroupVolumeConnection example.

Below shows what all of these elements look like in the Pure Storage Web Management tool. The first cmdlet, `New-PfaHostVolumeConnection` is used to connect the TEST-VOL1 volume to the new TEST-HOST1 that was created in Step 1. Next, `New-PfaHostGroupVolumeConnection` is used to connect the TEST-VOL2 volume to the TEST-HOSTGROUP created in Step 2.

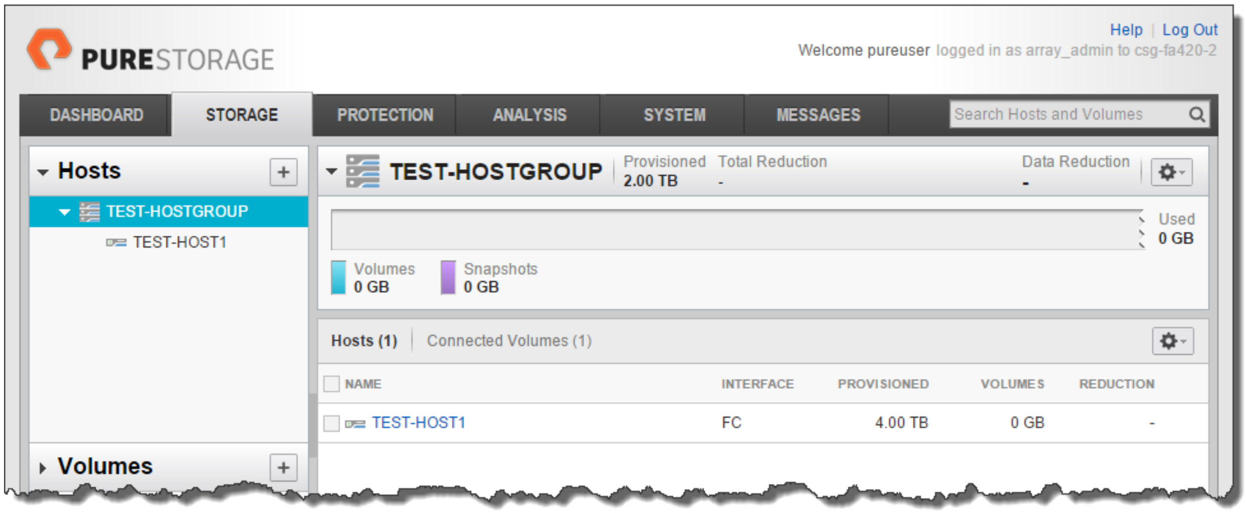


TEST-HOST1 — Provisioned 4.00 TB, Total Reduction -, Data Reduction -, Used 0 GB, Volumes 0 GB, Snapshots 0 GB

Connected Volumes (2) | Host Ports (2) | Details (0)

NAME	LUN	PROVISIONED	VOLUMES	SNAPSHOTS	REDUCTION
TEST-VOL1	1	2 TB	0 GB	0 GB	-
TEST-VOL2	10	2 TB	0 GB	0 GB	-

Figure 30. FlashArray Web Management view of the new host.



TEST-HOSTGROUP — Provisioned 2.00 TB, Total Reduction -, Data Reduction -, Used 0 GB, Volumes 0 GB, Snapshots 0 GB

Hosts (1) | Connected Volumes (1)

NAME	INTERFACE	PROVISIONED	VOLUMES	REDUCTION
TEST-HOST1	FC	4.00 TB	0 GB	-

Figure 31. FlashArray Web Management view of the new host group.

FlashRecover Snapshots

Previously we created 10 new 2TB volumes. Now that there are volumes, a host, and host group we can begin taking FlashRecover Snapshots of the volumes and show how easy it is leverage this Purity Operating Environment feature.

The next set of tasks to complete will be the following:

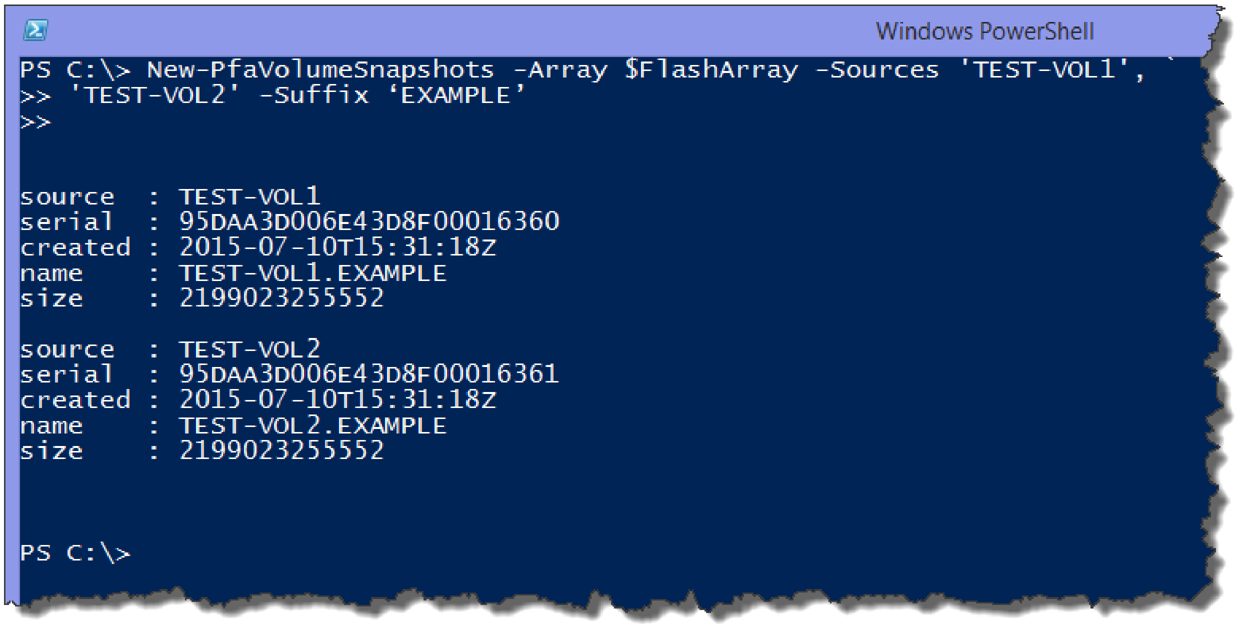
1. Create a snapshot of the TEST-VOL1 and TEST-VOL2 volumes.
2. Create a new volume from the TEST-VOL1 snapshot.
3. Overwrite the TEST-VOL2 volume with the newly created TEST-VOL2 snapshot.

Here are the steps to accomplish the above tasks:

Step 1

The following script uses the `New-PfaVolumeSnapshots` cmdlet to take a snapshot of two volumes; this cmdlet can be used with multiple volumes.

```
New-PfaVolumeSnapshots -Array $FlashArray -Sources 'TEST-VOL1', `
'TEST-VOL2' -Suffix 'EXAMPLE'
```



```
PS C:\> New-PfaVolumeSnapshots -Array $FlashArray -Sources 'TEST-VOL1', `
>> 'TEST-VOL2' -Suffix 'EXAMPLE'
>>


source  : TEST-VOL1
serial  : 95DAA3D006E43D8F00016360
created : 2015-07-10T15:31:18Z
name    : TEST-VOL1.EXAMPLE
size    : 2199023255552

source  : TEST-VOL2
serial  : 95DAA3D006E43D8F00016361
created : 2015-07-10T15:31:18Z
name    : TEST-VOL2.EXAMPLE
size    : 2199023255552



PS C:\>
```

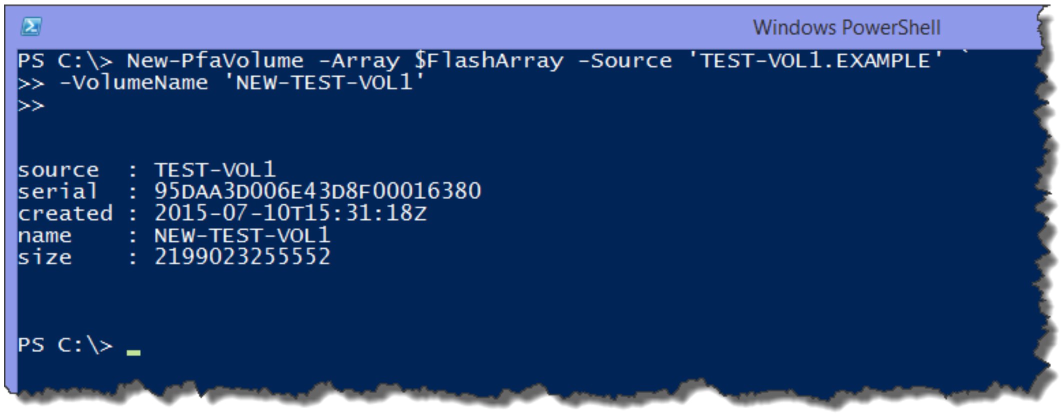
Figure 32. New-PfaVolumeSnapshots example.

Now these snapshots can be used to create new volumes, as we will do in Step 2, but they can also be used to restore (or overwrite) the parent volume with the snapshot. By restoring the volume with the snapshot, the volume is restored back to the point-in-time that the snapshot was taken. The snapshot is a crash consistent snapshot, which simply means that the data on the snapshot is in the same state it would be in if a host crashed (e.g. Lost power).

Step 2

The first task in Step 2 is to take the TEST-VOL1.EXAMPLE snapshot and create a new volume that can be connected to a host or host group.

```
New-PfaVolume -Array $FlashArray -Source 'TEST-VOL1.EXAMPLE' `
-VolumeName 'NEW-TEST-VOL1'
```



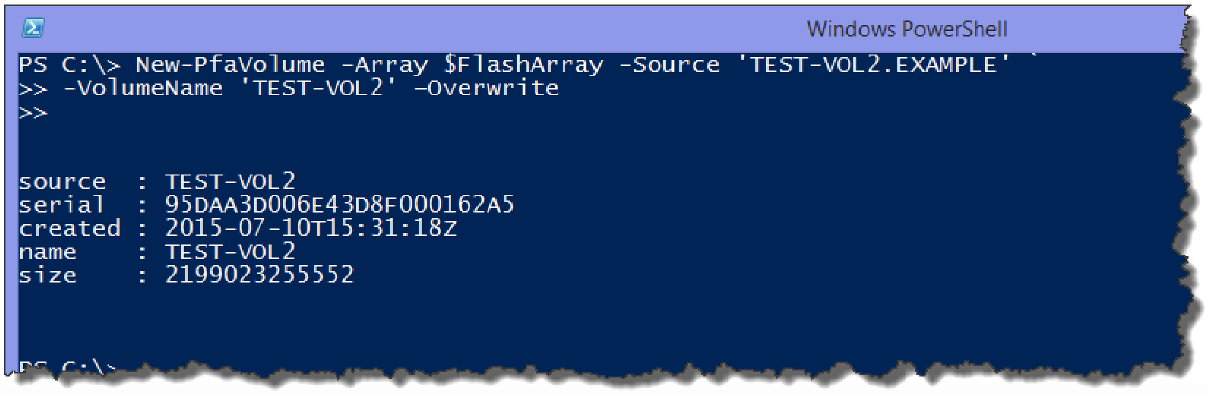
```
Windows PowerShell
PS C:\> New-PfaVolume -Array $FlashArray -Source 'TEST-VOL1.EXAMPLE' `
>> -VolumeName 'NEW-TEST-VOL1'
>>


source  : TEST-VOL1
serial  : 95DAA3D006E43D8F00016380
created : 2015-07-10T15:31:18Z
name    : NEW-TEST-VOL1
size    : 2199023255552


PS C:\> _
```

Figure 33. New-PfaVolume from Snapshot.

To connect the new volume to a host or host group, follow the steps in the previous section entitled Volume and Host Connections.

The next task is to take the TEST-VOL2.EXAMPLE snapshot and restore the TEST-VOL2 volume (the parent) with the snapshot.

```
New-PfaVolume -Array $FlashArray -Source 'TEST-VOL2.EXAMPLE' `
-VolumeName 'TEST-VOL2' -Overwrite
```



```
Windows PowerShell
PS C:\> New-PfaVolume -Array $FlashArray -Source 'TEST-VOL2.EXAMPLE' `
>> -VolumeName 'TEST-VOL2' -Overwrite
>>


source  : TEST-VOL2
serial  : 95DAA3D006E43D8F000162A5
created : 2015-07-10T15:31:18Z
name    : TEST-VOL2
size    : 2199023255552


PS C:\>
```

Figure 34. New-PfaVolume using Overwrite.

Protection Groups

So far we have created volumes, hosts, host groups and snapshots. The ability to take volume level snapshots using `New-PfaVolumeSnapshots` is a key data protection capability. Up to this point, we have demonstrated the process to create individual volume snapshots. In this section, we will explore doing this at scale using protection groups that contain multiple volumes, a host or host group. Protection groups offer both local snapshot retention management and scheduling as well as remote replication scheduling. This quick start guide is focused on local snapshot management.

The next set of tasks to complete will be the following:

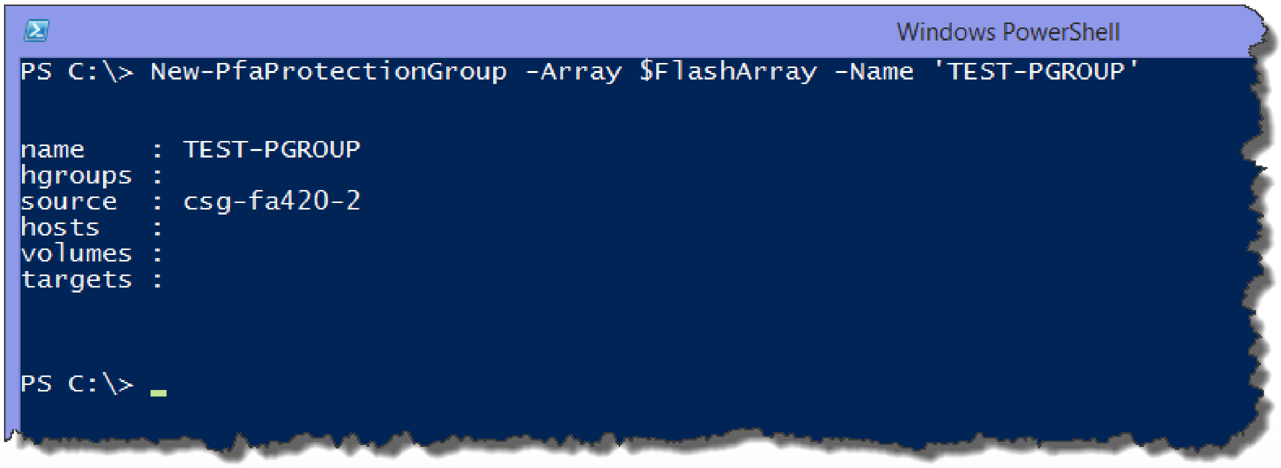
1. Create a new protection group, named TEST-PGROUP.
2. Add TEST-VOL1 – TEST-VOL10 volumes to TEST-PGROUP.
3. Manually create a protection group snapshot of TEST-PGROUP.
4. View TEST-PGROUP snapshot(s).
5. Set retention management policy for TEST-PGROUP and view TEST-PGROUP snapshot based on retention management policy.
6. Remove volumes from TEST-PGROUP then add host and host groups.

Here are the steps to accomplish the above tasks:

Step 1

Creating a new protection group is just as simple as creating a volume, host or host group. For now we are just creating the TEST-PGROUP and not using any of the other parameters (`–Volumes`, `–Hosts` or `–HostGroups`).

```
New-PfaProtectionGroup -Array $FlashArray -Name 'TEST-PGROUP'
```



```
PS C:\> New-PfaProtectionGroup -Array $FlashArray -Name 'TEST-PGROUP'


name     : TEST-PGROUP
hgroups  :
source   : csg-fa420-2
hosts    :
volumes  :
targets  :



PS C:\> _
```

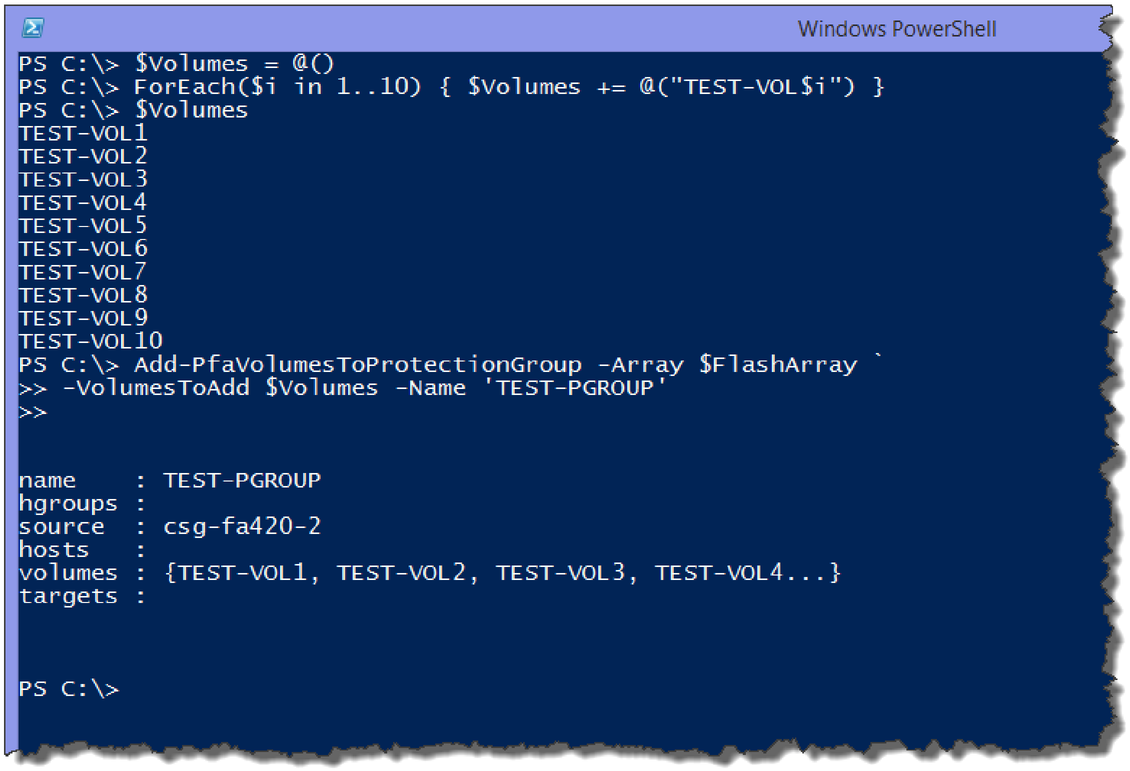
Figure 35. New-PfaProtectionGroup example.

Step 2

Next we will add volumes to the protection group using the `-Volumes` parameter. This parameter allows for passing multiple volume names but to keep the cmdlet cleaner we have created a simple array and added the TEST-VOL1 – TEST-VOL10 volumes which we will pass into the cmdlet.

```
$Volumes = @()
ForEach($i in 1..10) { $Volumes += @("TEST-VOL$i") }
$Volumes
Add-PfaVolumesToProtectionGroup -Array $FlashArray `
-VolumesToAdd $Volumes -Name 'TEST-PGROUP'
```

The first line in the script sample above created an empty array. Using a simple `ForEach` loop we fill the array with TEST1 – TEST10 volumes. Then the `$Volumes` array is checked for contents and then lastly we add the volumes to the TEST-PGROUP.



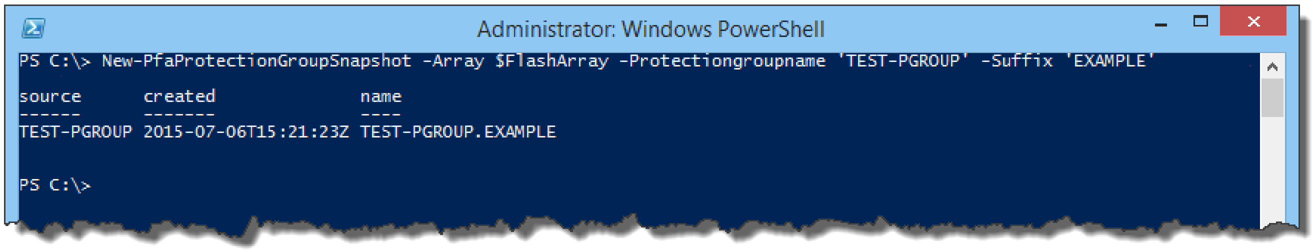
```
PS C:\> $Volumes = @()
PS C:\> ForEach($i in 1..10) { $Volumes += @("TEST-VOL$i") }
PS C:\> $Volumes
TEST-VOL1
TEST-VOL2
TEST-VOL3
TEST-VOL4
TEST-VOL5
TEST-VOL6
TEST-VOL7
TEST-VOL8
TEST-VOL9
TEST-VOL10
PS C:\> Add-PfaVolumesToProtectionGroup -Array $FlashArray `
>> -VolumesToAdd $Volumes -Name 'TEST-PGROUP'
>>


name     : TEST-PGROUP
hgroups  :
source   : csg-fa420-2
hosts    :
volumes  : {TEST-VOL1, TEST-VOL2, TEST-VOL3, TEST-VOL4...}
targets  :



PS C:\>
```

Figure 36. Add-PfaVolumesToProtectionGroup example.

Step 3

Now we have a protection group, TEST-PGROUP, with 10 volumes. In this step we will take a manual snapshot of the TEST-PGROUP. If the `-Suffix` parameter is not used, a numeric suffix will be created by the system.

```
New-PfaProtectionGroupSnapshot -Array $FlashArray `
-Protectiongroupname 'TEST-PGROUP' -Suffix 'EXAMPLE'
```



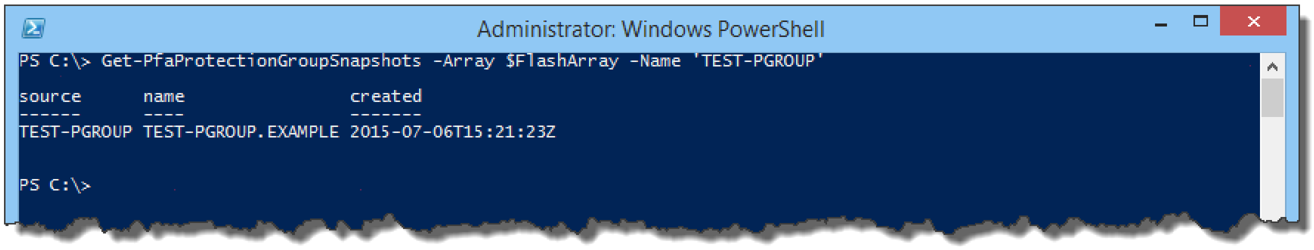
```
PS C:\> New-PfaProtectionGroupSnapshot -Array $FlashArray -Protectiongroupname 'TEST-PGROUP' -Suffix 'EXAMPLE'

source      created              name
------      -------              ----
TEST-PGROUP 2015-07-06T15:21:23Z TEST-PGROUP.EXAMPLE

PS C:\>
```

Figure 37. New-PfaProtectionGroupSnapshot example.

Step 4

We can now view the details of the newly created snapshot TEST-PGROUP.

```
Get-PfaProtectionGroupSnapshots -Array $FlashArray -Name 'TEST-PGROUP'
```



```
PS C:\> Get-PfaProtectionGroupSnapshots -Array $FlashArray -Name 'TEST-PGROUP'

source      name                created
------      ----                -------
TEST-PGROUP TEST-PGROUP.EXAMPLE 2015-07-06T15:21:23Z

PS C:\>
```

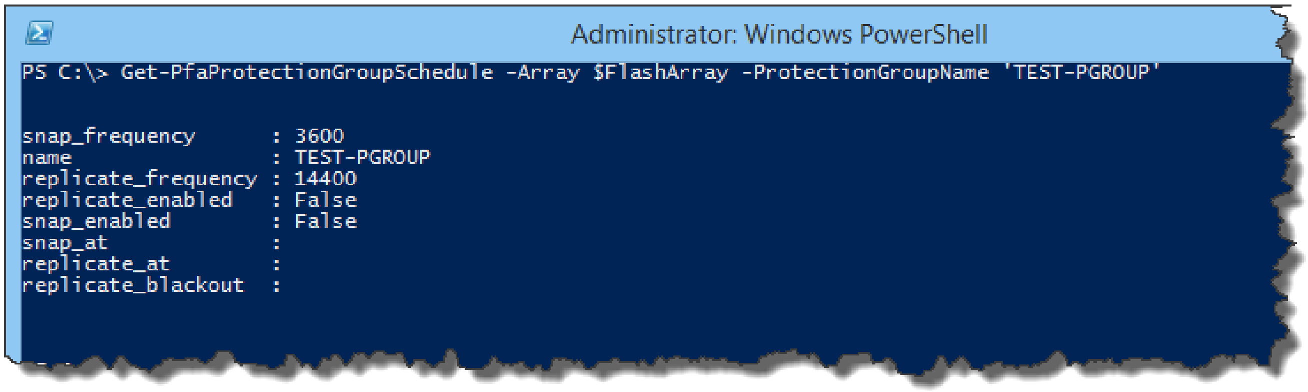
Figure 38. Get-PfaProtectionGroupSnapshots example.

Step 5

We created a snapshot of a protection group manually in Step 4. In this step we will show how to setup the local snapshot schedule. It is important to first provide a view of how the Snapshot and Replication Schedule appear in the Pure Storage Web Management tool in Figure 40. We will be focusing on modifying the Snapshot Schedule. Figure 40 illustrates a schedule to take a snapshot every 1 hour. When working with schedules in the PowerShell SDK they are shown in seconds. Example, 1 hour = 3600 seconds. To demonstrate this try the following script:

```
Get-PfaProtectionGroupSchedule -Array $FlashArray `
-ProtectionGroupName 'TEST-PGROUP'
```

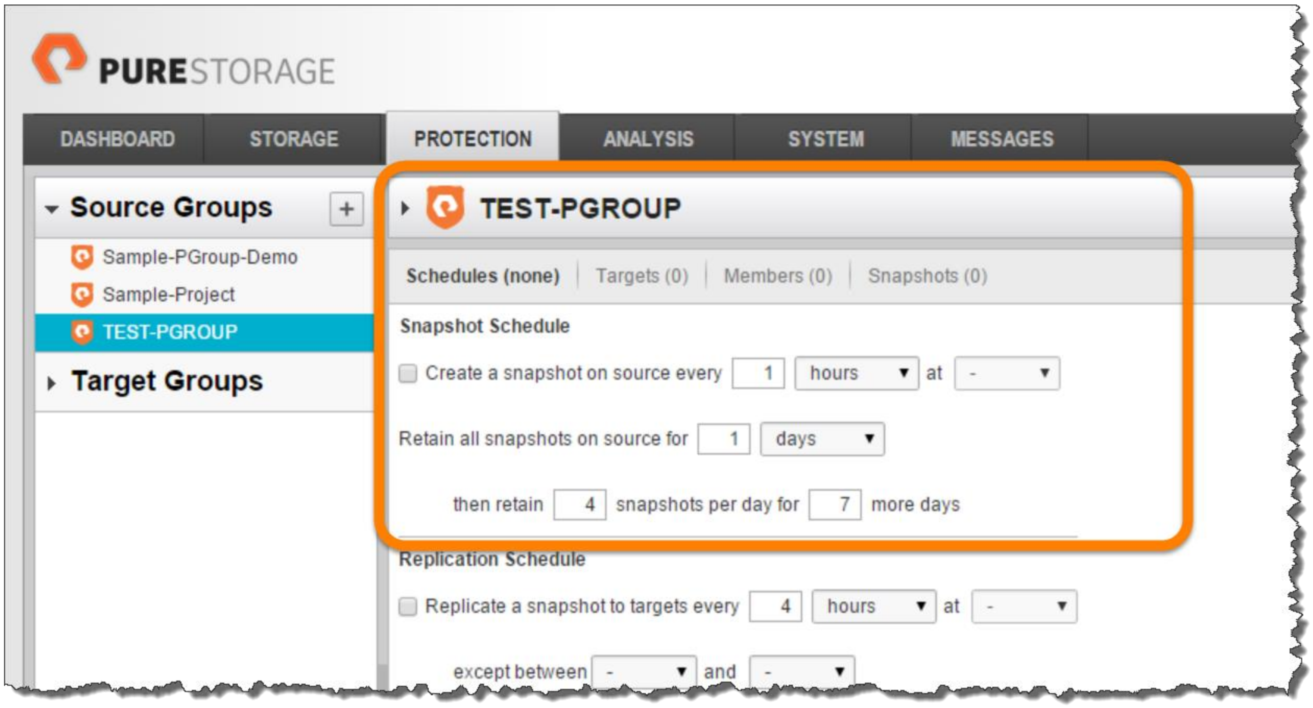
Figure 39 shows the same output seen in Figure 40.



Administrator: Windows PowerShell

```
PS C:\> Get-PfaProtectionGroupSchedule -Array $FlashArray -ProtectionGroupName 'TEST-PGROUP'


snap_frequency      : 3600
name                : TEST-PGROUP
replicate_frequency : 14400
replicate_enabled   : False
snap_enabled        : False
snap_at             :
replicate_at        :
replicate_blackout  :
```

Figure 39. Get-PfaProtectionGroupSchedule example.



Snapshot Schedule

☐ Create a snapshot on source every 1 hours at -

Retain all snapshots on source for 1 days

then retain 4 snapshots per day for 7 more days

Replication Schedule

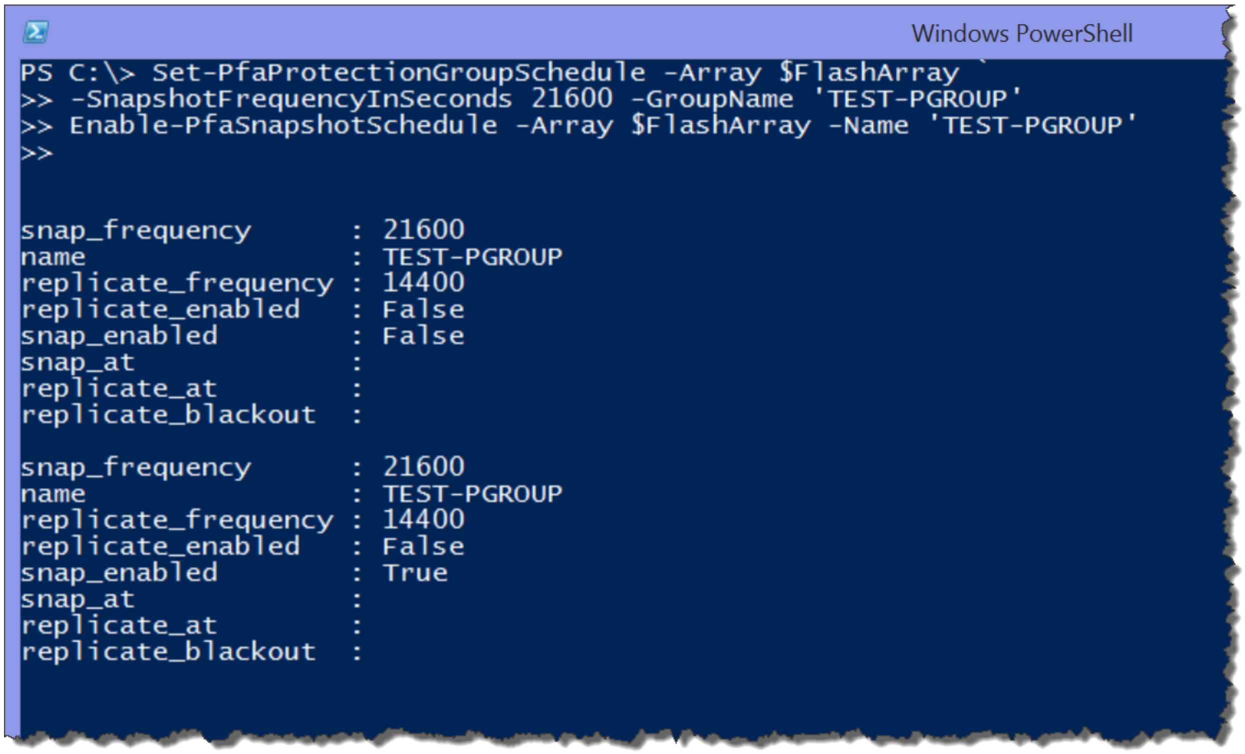
☐ Replicate a snapshot to targets every 4 hours at -

except between - and -

Figure 40. TEST-PGROUP schedules for snapshot and replication.

The next step is to set a new snapshot schedule for 6 hours or 21,600 seconds.

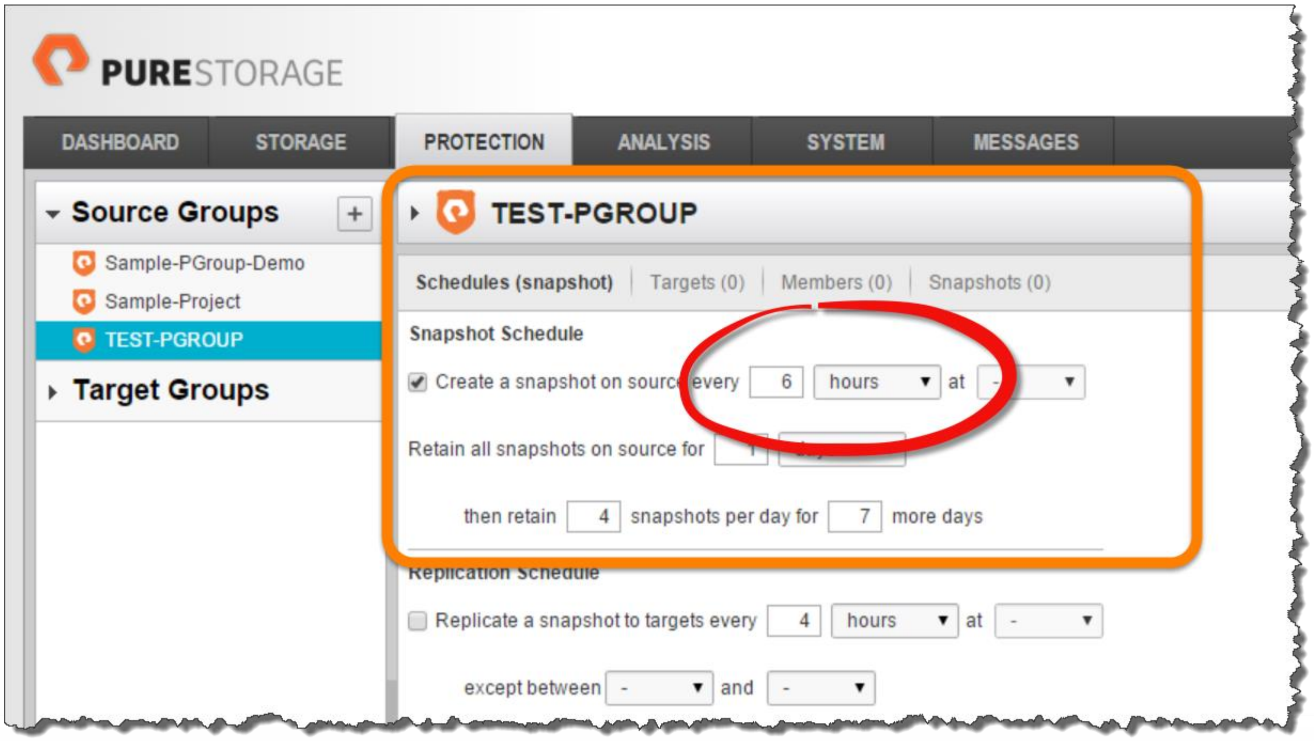
```
Set-PfaProtectionGroupSchedule -Array $FlashArray `
-SnapshotFrequencyInSeconds 21600 -GroupName 'TEST-PGROUP'
Enable-PfaSnapshotSchedule -Array $FlashArray -Name 'TEST-PGROUP'
```



```
PS C:\> Set-PfaProtectionGroupSchedule -Array $FlashArray `
>> -SnapshotFrequencyInSeconds 21600 -GroupName 'TEST-PGROUP'
>> Enable-PfaSnapshotSchedule -Array $FlashArray -Name 'TEST-PGROUP'
>>


snap_frequency       : 21600
name                 : TEST-PGROUP
replicate_frequency  : 14400
replicate_enabled    : False
snap_enabled         : False
snap_at              :
replicate_at         :
replicate_blackout   :

snap_frequency       : 21600
name                 : TEST-PGROUP
replicate_frequency  : 14400
replicate_enabled    : False
snap_enabled         : True
snap_at              :
replicate_at         :
replicate_blackout   :
```

Figure 41. Set-PfaProtectionGroupSchedule example.

Figure 42 shows the update to 6 hours in the Pure Storage Web Management tool.



Snapshot Schedule

☑ Create a snapshot on source every 6 hours at -

Retain all snapshots on source for 1

then retain 4 snapshots per day for 7 more days

Replication Schedule

☐ Replicate a snapshot to targets every 4 hours at -

except between - and -

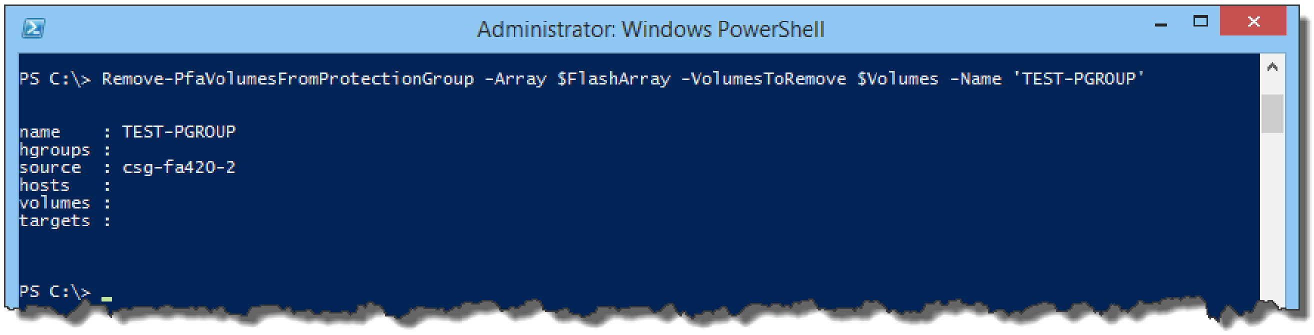
Figure 42. Updated Snapshot Schedule for TEST-PGROUP.

Step 6

In this final step we are going to remove the volumes added previously and then add a single host, then lastly remove the host and add a host group. The host and host group used in this section are the ones previously created, TEST-HOST1 and TEST-HOSTGROUP, respectively. As mentioned earlier, a protection group can provide volume protection for one or many volumes, hosts or host groups. If we look at the different options at a granularity level, protecting volumes is the most granular. Protecting a host(s) means any object (volume) connected to that host(s) is protected. The same principle applies to the host group, any object (hosts) that are connected along with the objects that are connected to the individual hosts (volumes) are protected. Any protection group snapshot is an atomic operation providing consistent timestamps across all objects.

The below script is reusing the previously created `$Volumes` array from Step 2.

```
$Volumes = @()
ForEach($i in 1..10) { $Volumes += @("TEST-VOL$i") }
$Volumes
Remove-PfaVolumesFromProtectionGroup -Array $FlashArray `
-VolumesToRemove $Volumes -Name 'TEST-PGROUP'
```

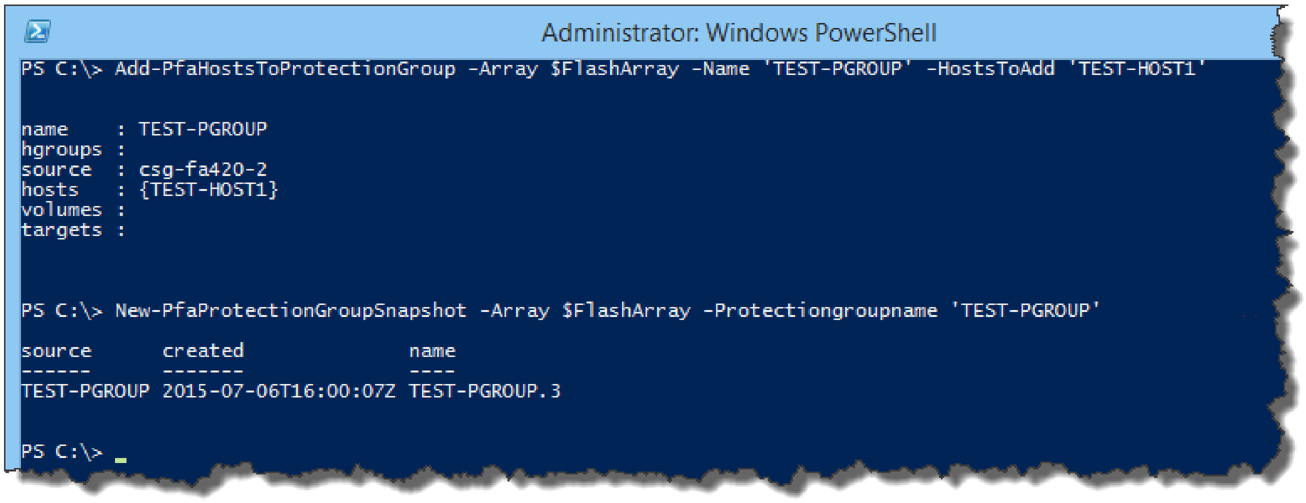


```
PS C:\> Remove-PfaVolumesFromProtectionGroup -Array $FlashArray -VolumesToRemove $Volumes -Name 'TEST-PGROUP'


name    : TEST-PGROUP
hgroups :
source  : csg-fa420-2
hosts   :
volumes :
targets :


PS C:\> _
```

Figure 43. Remove-PfaVolumesFromProtectionGroup example.

The next step is to add the TEST-HOST1 to the TEST-PGROUP and then take a new snapshot. In this example the process of taking a new snapshot does not use the `-Suffix` parameter and show the TEST-PGROUP.3 as the resultant snapshot name.

```
Add-PfaHostsToProtectionGroup -Array $FlashArray -Name 'TEST-PGROUP' `
-HostsToAdd 'TEST-HOST1'
New-PfaProtectionGroupSnapshot -Array $FlashArray `
-Protectiongroupname 'TEST-PGROUP'
```

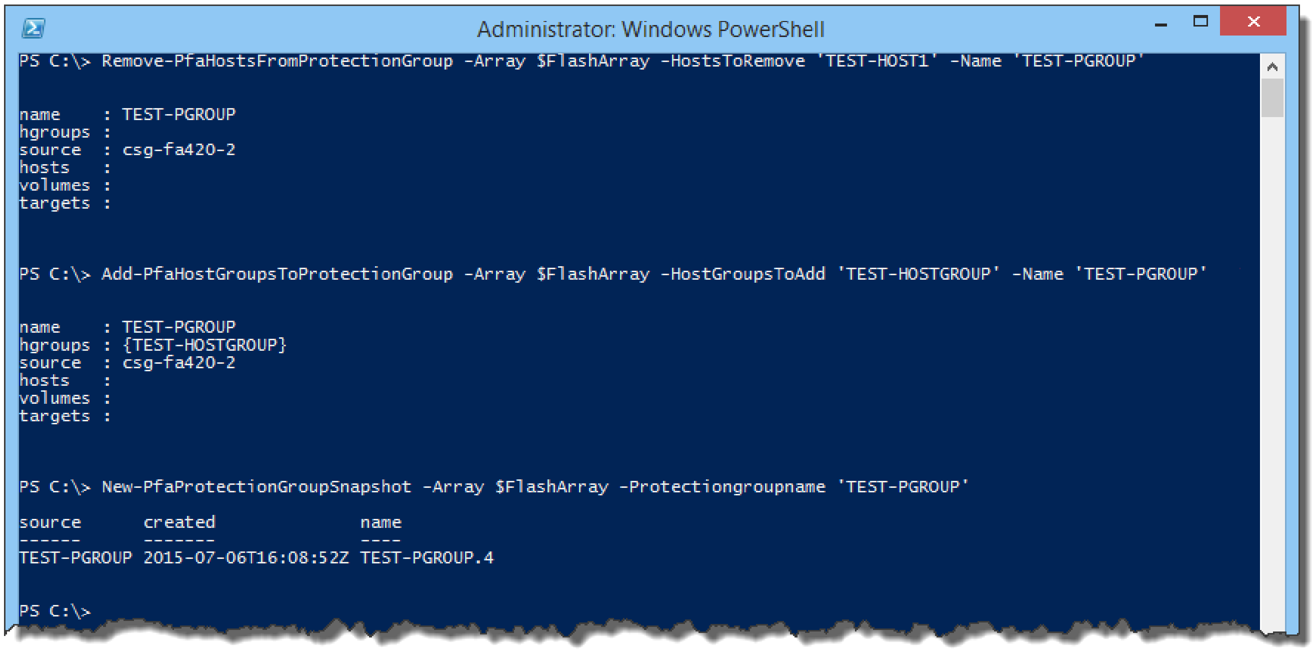


```
PS C:\> Add-PfaHostsToProtectionGroup -Array $FlashArray -Name 'TEST-PGROUP' -HostsToAdd 'TEST-HOST1'


name     : TEST-PGROUP
hgroups  :
source   : csg-fa420-2
hosts    : {TEST-HOST1}
volumes  :
targets  :



PS C:\> New-PfaProtectionGroupSnapshot -Array $FlashArray -Protectiongroupname 'TEST-PGROUP'

source      created              name
------      -------              ----
TEST-PGROUP 2015-07-06T16:00:07Z TEST-PGROUP.3


PS C:\> _
```

Figure 44. Add-PfaHostsToProtectionGroup and New-PfaProtectionGroupSnapshot.

A similar cmdlet as used in Step 6 is now used to remove the host from the protection group and then add the TEST-HOSTGROUP to the protection group, TEST-PGROUP.

```
Remove-PfaHostsFromProtectionGroup -Array $FlashArray `
-HostsToRemove 'TEST-HOST1' -Name 'TEST-PGROUP'
Add-PfaHostGroupsToProtectionGroup -Array $FlashArray `
-HostGroupsToAdd 'TEST-HOSTGROUP' -Name 'TEST-PGROUP'
New-PfaProtectionGroupSnapshot -Array $FlashArray `
-Protectiongroupname 'TEST-PGROUP'
```



```
PS C:\> Remove-PfaHostsFromProtectionGroup -Array $FlashArray -HostsToRemove 'TEST-HOST1' -Name 'TEST-PGROUP'


name     : TEST-PGROUP
hgroups  :
source   : csg-fa420-2
hosts    :
volumes  :
targets  :



PS C:\> Add-PfaHostGroupsToProtectionGroup -Array $FlashArray -HostGroupsToAdd 'TEST-HOSTGROUP' -Name 'TEST-PGROUP'


name     : TEST-PGROUP
hgroups  : {TEST-HOSTGROUP}
source   : csg-fa420-2
hosts    :
volumes  :
targets  :



PS C:\> New-PfaProtectionGroupSnapshot -Array $FlashArray -Protectiongroupname 'TEST-PGROUP'

source      created              name
------      -------              ----
TEST-PGROUP 2015-07-06T16:08:52Z TEST-PGROUP.4


PS C:\>
```

Figure 45. Removing, Adding and creating new snapshot for a Protection Group.

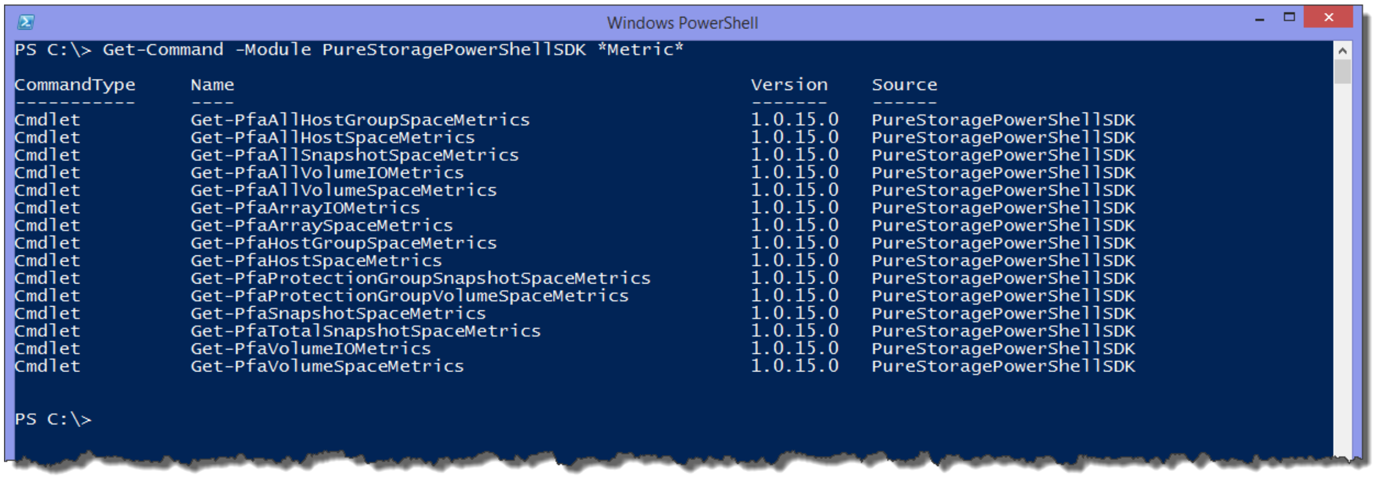
Monitor Metrics

One of the most helpful operations that can be performed on the FlashArray is monitoring the various detailed metrics the Purity Operating Environment provides. The SDK provides 15 different cmdlets that can be used to monitor specific objects.

Using the following command you can get a list of all the different cmdlets that show either Space or I/O metrics.

```
Get-Command -Module PureStoragePowerShellSDK *Metric*
```

A list of those cmdlets is as follows:



```
PS C:\> Get-Command -Module PureStoragePowerShellSDK *Metric*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Get-PfaAllHostGroupSpaceMetrics                    1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllHostSpaceMetrics                         1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllSnapshotSpaceMetrics                     1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllVolumeIOMetrics                          1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaAllVolumeSpaceMetrics                       1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaArrayIOMetrics                              1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaArraySpaceMetrics                           1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaHostGroupSpaceMetrics                       1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaHostSpaceMetrics                            1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaProtectionGroupSnapshotSpaceMetrics         1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaProtectionGroupVolumeSpaceMetrics           1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaSnapshotSpaceMetrics                        1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaTotalSnapshotSpaceMetrics                   1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaVolumeIOMetrics                             1.0.15.0   PureStoragePowerShellSDK
Cmdlet          Get-PfaVolumeSpaceMetrics                          1.0.15.0   PureStoragePowerShellSDK


PS C:\>
```

Figure 46. List of SDK cmdlets for retrieving metrics.

The next task to complete will be the following:

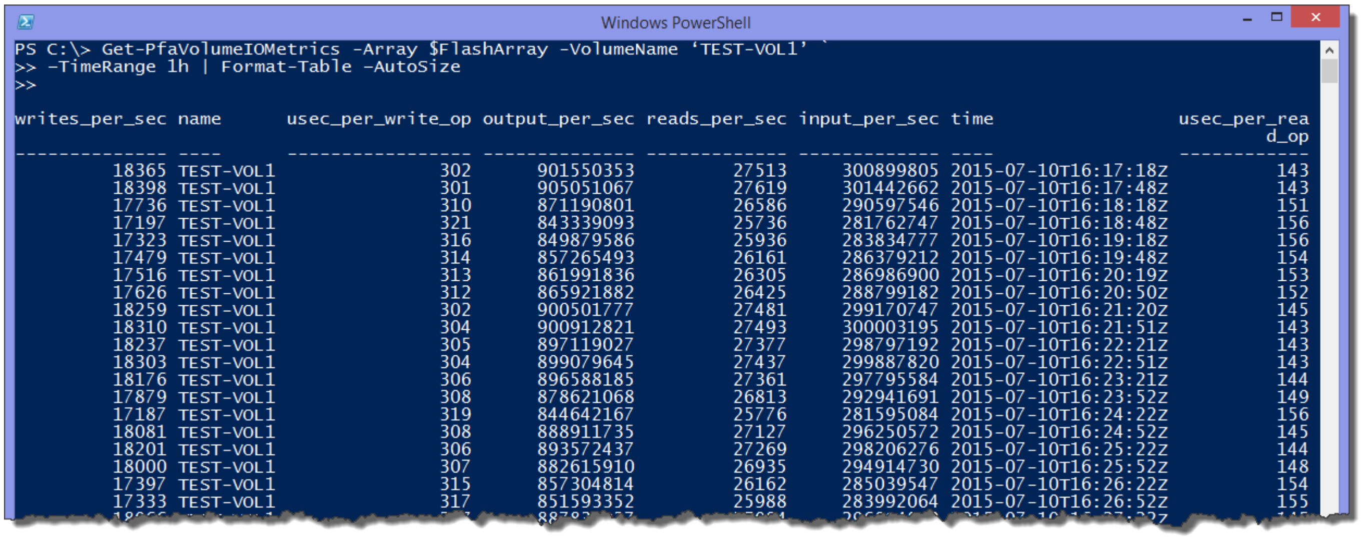
1. Retrieve the IO Metrics for a specific Volume

Here is the step to complete the above task:

Step 1

For this quick start guide we will show the use of `Get-PfaVolumeIOMetrics` for the TEST-VOL1 volume over a 1 hour time range.

```
Get-PfaVolumeIOMetrics -Array $FlashArray -VolumeName 'TEST-VOL1' `
-TimeRange 1h | Format-Table -AutoSize
```



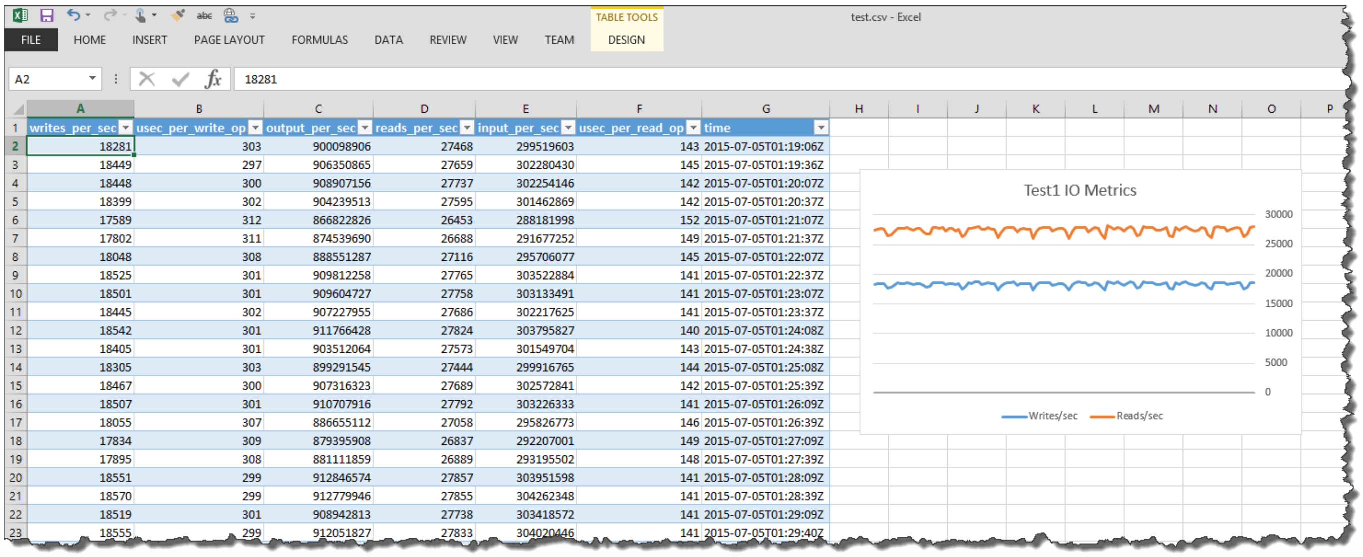
```
PS C:\> Get-PfaVolumeIOMetrics -Array $FlashArray -VolumeName 'TEST-VOL1' `
>> -TimeRange 1h | Format-Table -AutoSize
>>

writes_per_sec name      usec_per_write_op output_per_sec reads_per_sec input_per_sec time                  usec_per_rea
                                                                                                                    d_op
-------------- ----      ----------------- -------------- ------------- ------------- ----                  ------------
         18365 TEST-VOL1               302      901550353         27513     300899805 2015-07-10T16:17:18Z           143
         18398 TEST-VOL1               301      905051067         27619     301442662 2015-07-10T16:17:48Z           143
         17736 TEST-VOL1               310      871190801         26586     290597546 2015-07-10T16:18:18Z           151
         17197 TEST-VOL1               321      843339093         25736     281762747 2015-07-10T16:18:48Z           156
         17323 TEST-VOL1               316      849879586         25936     283834777 2015-07-10T16:19:18Z           156
         17479 TEST-VOL1               314      857265493         26161     286379212 2015-07-10T16:19:48Z           154
         17516 TEST-VOL1               313      861991836         26305     286986900 2015-07-10T16:20:19Z           153
         17626 TEST-VOL1               312      865921882         26425     288799182 2015-07-10T16:20:50Z           152
         18259 TEST-VOL1               302      900501777         27481     299170747 2015-07-10T16:21:20Z           145
         18310 TEST-VOL1               304      900912821         27493     300003195 2015-07-10T16:21:51Z           143
         18237 TEST-VOL1               305      897119027         27377     298797192 2015-07-10T16:22:21Z           143
         18303 TEST-VOL1               304      899079645         27437     299887820 2015-07-10T16:22:51Z           143
         18176 TEST-VOL1               306      896588185         27361     297795584 2015-07-10T16:23:21Z           144
         17879 TEST-VOL1               308      878621068         26813     292941691 2015-07-10T16:23:52Z           149
         17187 TEST-VOL1               319      844642167         25776     281595084 2015-07-10T16:24:22Z           156
         18081 TEST-VOL1               308      888911735         27127     296250572 2015-07-10T16:24:52Z           145
         18201 TEST-VOL1               306      893572437         27269     298206276 2015-07-10T16:25:22Z           144
         18000 TEST-VOL1               307      882615910         26935     294914730 2015-07-10T16:25:52Z           148
         17397 TEST-VOL1               315      857304814         26162     285039547 2015-07-10T16:26:22Z           154
         17333 TEST-VOL1               317      851593352         25988     283992064 2015-07-10T16:26:52Z           155
```

Figure 47. Get-PfaVolumeIOMetrics example.

Figure 47 shows all of the output values for the TEST-VOL1 volume. It is nice to see these values in the Windows PowerShell console but many administrators wish to graph metrics to get more insight into performance.

```
Get-PfaVolumeIOMetrics -Array $FlashArray -VolumeName 'TEST-VOL1' `
-TimeRange 1h | Export-Csv -Path 'C:\temp\test.csv'
```

The above line of script takes the same output and passes it through the cmdlet pipeline to the `Export-Csv` cmdlet to create the test.csv file. This test.csv file can then be opened in Microsoft Excel and manipulated in a number of ways. The screenshot below shows an example of a line graph for write/sec and read/sec in Microsoft Excel.

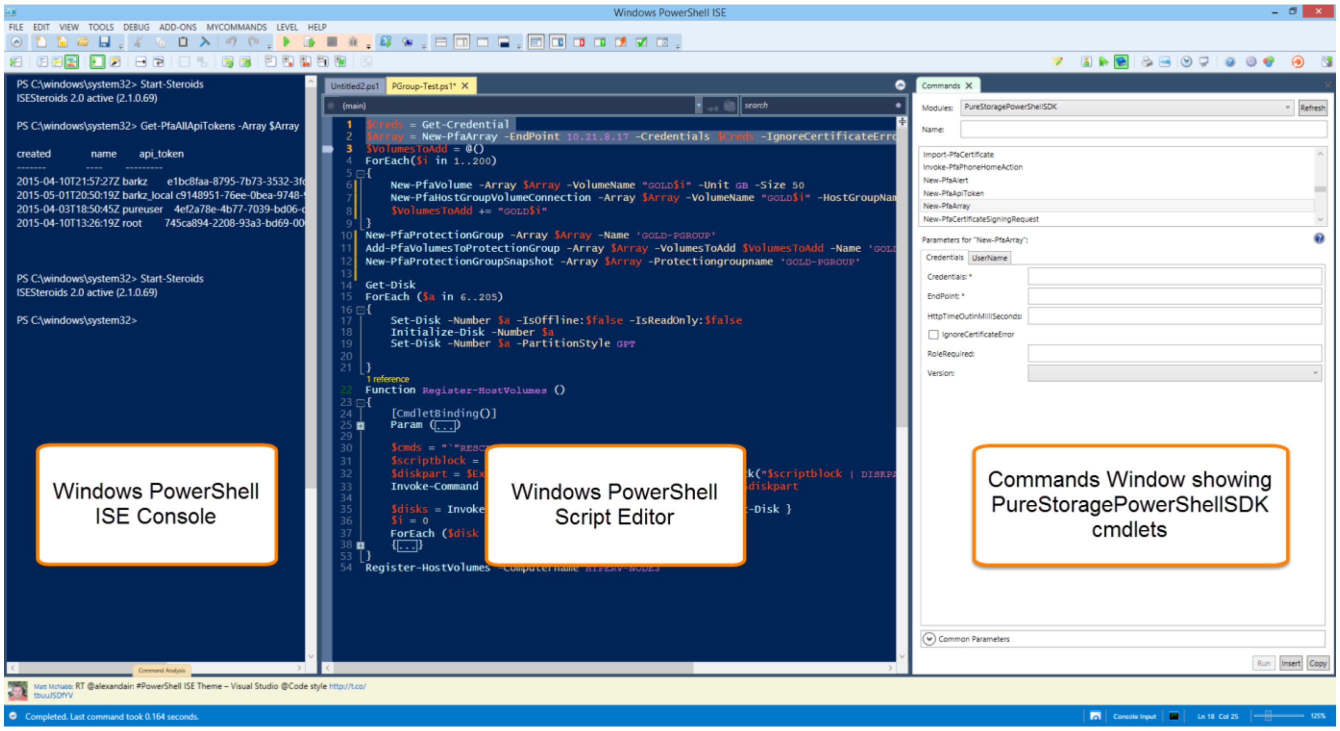


writes_per_sec	usec_per_write_op	output_per_sec	reads_per_sec	input_per_sec	usec_per_read_op	time
18281	303	900098906	27468	299519603	143	2015-07-05T01:19:06Z
18449	297	906350865	27659	302280430	145	2015-07-05T01:19:36Z
18448	300	908907156	27737	302254146	142	2015-07-05T01:20:07Z
18399	302	904239513	27595	301462869	142	2015-07-05T01:20:37Z
17589	312	866822826	26453	288181998	152	2015-07-05T01:21:07Z
17802	311	874539690	26688	291677252	149	2015-07-05T01:21:37Z
18048	308	888551287	27116	295706077	145	2015-07-05T01:22:07Z
18525	301	909812258	27765	303522884	141	2015-07-05T01:22:37Z
18501	301	909604727	27758	303133491	141	2015-07-05T01:23:07Z
18445	302	907227955	27686	302217625	141	2015-07-05T01:23:37Z
18542	301	911766428	27824	303795827	140	2015-07-05T01:24:08Z
18405	301	903512064	27573	301549704	143	2015-07-05T01:24:38Z
18305	303	899291545	27444	299916765	144	2015-07-05T01:25:08Z
18467	300	907316323	27689	302572841	142	2015-07-05T01:25:39Z
18507	301	910707916	27792	303226333	141	2015-07-05T01:26:09Z
18055	307	886655112	27058	295826773	146	2015-07-05T01:26:39Z
17834	309	879395908	26837	292207001	149	2015-07-05T01:27:09Z
17895	308	881111859	26889	293195502	148	2015-07-05T01:27:39Z
18551	299	912846574	27857	303951598	141	2015-07-05T01:28:09Z
18570	299	912779946	27855	304262348	141	2015-07-05T01:28:39Z
18519	301	908942813	27738	303418572	141	2015-07-05T01:29:09Z
18555	299	912051827	27833	304020446	141	2015-07-05T01:29:40Z

Figure 48. Imported test.csv in Microsoft Excel.

Windows PowerShell Integrated Scripting Environment (ISE)

All of the examples provided in this quick start guide focused on using the Windows PowerShell command shell. The simple and straightforward way of working with PowerShell to run cmdlets. There is also another tool that comes with Microsoft Windows by default called the Windows PowerShell Integrated Scripting Environment (ISE). The ISE is an immensely valuable tool for anyone working on larger and more complicated scripts. The ISE provides script editing, debugging and the Command Window for help with modules and associated cmdlets. The basic features of ISE are shown in the screenshot below.


Windows PowerShell ISE
FILE EDIT VIEW TOOLS DEBUG ADD-ONS MYCOMMANDS LEVEL HELP
Windows PowerShell ISE Console
Untitled2.ps1
PGroup-Test.ps1*
(main)
search
Windows PowerShell Script Editor
Commands
Commands Window showing PureStoragePowerShellSDK cmdlets
Common Parameters
Run
Insert
Copy
Completed. Last command took 0.164 seconds.

```
PS C:\windows\system32> Start-Steroids
ISESteroids 2.0 active (2.1.0.69)

PS C:\windows\system32> Get-PfaAllApiTokens -Array $Array

created              name        api_token
-------              ----        ---------
2015-04-10T21:57:27Z barkz       e1bc8faa-8795-7b73-3532-3f
2015-05-01T20:50:19Z barkz_local c9148951-76ee-0bea-9748-
2015-04-03T18:50:45Z pureuser    4ef2a78e-4b77-7039-bd06-
2015-04-10T13:26:19Z root        745ca894-2208-93a3-bd69-00



PS C:\windows\system32> Start-Steroids
ISESteroids 2.0 active (2.1.0.69)

PS C:\windows\system32>
```

```
$Creds = Get-Credential
$Array = New-PfaArray -EndPoint 10.21.8.17 -Credentials $Creds -IgnoreCertificateErro
$VolumesToAdd = @()
ForEach($i in 1..200)
{
    New-PfaVolume -Array $Array -VolumeName "GOLD$i" -Unit GB -Size 50
    New-PfaHostGroupVolumeConnection -Array $Array -VolumeName "GOLD$i" -HostGroupNam
    $VolumesToAdd += "GOLD$i"
}
New-PfaProtectionGroup -Array $Array -Name 'GOLD-PGROUP'
Add-PfaVolumesToProtectionGroup -Array $Array -VolumesToAdd $VolumesToAdd -Name 'GOLI
New-PfaProtectionGroupSnapshot -Array $Array -Protectiongroupname 'GOLD-PGROUP'

Get-Disk
ForEach ($a in 6..205)
{
    Set-Disk -Number $a -IsOffline:$false -IsReadOnly:$false
    Initialize-Disk -Number $a
    Set-Disk -Number $a -PartitionStyle GPT

}
1 reference
Function Register-HostVolumes ()
{
    [CmdletBinding()]
    Param (...)

    $cmds = "`"RESC
    $scriptblock =
    $diskpart = $Ex                     k("$scriptblock | DISKPA
    Invoke-Command                      $diskpart

    $disks = Invoke                     -Disk }
    $i = 0
    ForEach ($disk
    {...}
}
Register-HostVolumes -ComputerName HYPERV-NODE3
```

```
Modules: PureStoragePowerShellSDK
Name:

Import-PfaCertificate
Invoke-PfaPhoneHomeAction
New-PfaAlert
New-PfaApiToken
New-PfaArray
New-PfaCertificateSigningRequest

Parameters for "New-PfaArray":
Credentials | UserName
Credentials: *
EndPoint: *
HttpTimeOutInMilliSeconds:
[ ] IgnoreCertificateError
RoleRequired:
Version:
```

Figure 49. Windows PowerShell ISE.

Depending on whether you are using a Windows client or Windows Server you will need to do a search for PowerShell_ISE then you can add it to your taskbar, Start screen or desktop for quick access.

The Commands Window is very important to call out because it provides a way of viewing the different parameters for individual cmdlets. You can also enter them in the dynamically created form fields and then Run/Insert/Copy into a script that you are developing.

!

The Windows PowerShell ISE view above is using ISESteroids which provides an additional set of enhanced functionality to the PowerShell ISE. You can read more about ISESteroids at http://www.powertheshell.com/isesteroids

Summary

In this quick start guide we covered the following:

- A quick checklist for those of us who don't read install guides (don't worry I include myself in that group)
- Requirements of Windows PowerShell for the new Pure Storage PowerShell SDK
- Basic SDK operations for connecting, creating a volume, taking a snapshot and restoring that snapshot to a new volume or overwriting the existing volume.
- Brief discussion on the Windows PowerShell ISE

We hope the details in this quick start guide help you start to create your first reusable script to automate or manage your Pure Storage FlashArray. Lots more samples, help updates and additional quick start guides on the subjects of databases, hypervisors and the Pure Storage PowerShell Toolkit 3.0 are coming!

References

The following references contain valuable information related to Windows PowerShell.

2. Windows PowerShell Scripting - https://technet.microsoft.com/en-us/scriptcenter/powershell.aspx
3. Windows PowerShell Users Guide - https://technet.microsoft.com/en-us/library/cc196356.aspx
4. Getting Started with Windows PowerShell - https://technet.microsoft.com/library/hh857337.aspx
5. Pure Storage PowerShell Toolkit – https://github.com/barkz/PureStoragePowerShellToolkit
6. Windows 8 Packages - https://technet.microsoft.com/en-us/library/hh825549.aspx
7. Starting Windows PowerShell on Windows 8 and Windows - https://technet.microsoft.com/en-us/library/hh847889.aspx
8. PowerShell.org - http://powershell.org/wp/
9. ISE Steroids – http://www.powertheshell.com/isesteroid

About the Author

As a Solutions Architect, Barkz is creating the foundation knowledgebase for implementing Microsoft server technologies on Pure Storage. Those core items include best practices, reference architectures, management tasks, automation scripts and examples for application extensibility. With more than 20 years of experience with Microsoft solutions, Barkz has been part of all aspects from architecture, user design, development, test, release and administration. Barkz has experience in Windows PowerShell, Windows Server, Microsoft SQL Server (Admin & Development), Microsoft SharePoint Server (Admin & Development), Microsoft Hyper-V, Visual Studio, C# and REST API.

Barkz blog: http://www.purestorage.com/blog/author/barkz/

Twitter: @purepowershell

Demonstration Videos
YouTube

Pure1 Community
Programming Interfaces Community Page

Pure Storage PowerShell Toolkit
https://github.com/barkz/PureStoragePowerShellToolkit

PURE STORAGE

Pure Storage, Inc.
Twitter: @purestorage
www.purestorage.com

650 Castro Street, Suite #260
Mountain View, CA 94041

T: 650-290-6088
F: 650-625-9667

Sales: sales@purestorage.com
Support: support@purestorage.com
Media: pr@purestorage.com
General: info@purestorage.com