Technical Report

NetApp Disaster Recovery for the Microsoft Dynamic Data Center

Microsoft Alliance Engineering, NetApp
June 2010 | TR-3844

EXECUTIVE SUMMARY

This report provides technical guidance and code for cloud service providers who want to implement NetApp® disaster recovery for the Microsoft Dynamic Data Center, based on Microsoft® Windows Server® 2008 R2 Hyper-V™ and Microsoft System Center.

TABLE OF CONTENTS

1 INTRODUCTION

NetApp disaster recovery for the Microsoft Dynamic Data Center, based on Windows Server 2008 R2 Hyper-V, is designed to provide secure and highly reliable disaster recovery (DR) services that can easily scale as your customers' needs evolve. Our solutions leverage our extensive experience in providing efficient, enterprise-class storage solutions for Microsoft applications. Proven cloud service-delivery processes allow us to tailor your solution to include other business needs such as high-availability clustering, backup, and compliance, and to implement quickly to speed your time to revenue. We also help keep costs low for you and your customers with efficient solutions that reduce storage costs and streamline IT productivity.

Elements of our DR solutions include:

- Configurable, disk-based data replication
- Application-consistent NetApp Snapshot™ copies in physical and virtual environments
- Automated replication and failover integrated with Microsoft System Center
- Built-in compression to minimize network costs
- Integrated storage efficiency software to minimize storage, space, and energy costs
- Flexible cloning of data sets to perform risk-free, on-demand testing of disaster readiness

2 TEST DRIVE OUR CAPABILITIES

By using iSCSI LUNS containing Hyper-V VHDs, NetApp and Microsoft are able to offer a robust solution for Hyper-V wide area data protection. To help your engineering team get familiar with these capabilities, NetApp and Microsoft have supplied best practices and test code. The solution leverages the following NetApp and Microsoft technologies and products:

- Microsoft Windows Server 2008 R2 Datacenter/Hyper-V
- Microsoft System Center Operations Manager 2007 R2
- Microsoft System Center Virtual Machine Manager 2008 R2
- Microsoft Dynamic Data Center Toolkit for Hosters
- NetApp SnapMirror®
- NetApp SnapManager® for Hyper-V
- NetApp FlexClone®
- NetApp FASX000 systems

2.1 GETTING STARTED

At the heart of this disaster recovery solution is the replication efficiency of NetApp SnapMirror. To understand SnapMirror fundamentals and design and performance considerations, read Technical Report 3326:

http://media.netapp.com/documents/tr-3326.pdf

Assuming that the right software and hardware components are in place, the next step is to establish a SnapMirror relationship between the primary and secondary NetApp FAS controllers by using the scripts in section 2.2. When the relationship is established, writes to the primary controller are consistently sent to the remote NetApp SnapMirror volume. This enables a write-consistent state to be present for the Hyper-V VHD on the standby system, if needed. This solution can be further augmented by using SnapManager for Hyper-V to make consistent backups, making it possible to restore from multiple backups as needed.

The virtual infrastructure built on Hyper-V can be in a standalone or a clustered environment, depending on the availability requirements of the virtual machines and the applications that it hosts. Hyper-V setup with standalone servers is easy to build and manage, whereas the clustered setup is a bit complex and requires the user to be knowledgeable about Windows

Failover Clustering. A component failure (server hardware issues, storage connectivity issues) in a single Hyper-V server setup results in downtime for the entire infrastructure, whereas a clustered Hyper-V server can provide virtual machine availability by using the Live Migration feature of Windows 2008 R2.

2.2 SAMPLE TEST CODE

This section provides sample test code to perform failover of virtual machines hosted in a Hyper-V environment over Cluster Shared Volumes (CSVs) from the main site to the disaster recovery site. It includes test scripts written for Windows PowerShell™ to perform SnapMirror setup, virtual machine failover from the active site to the recovery site, and other supporting scripts. It also contains VBScripts to import and export the test code to configure virtual machines.

Use the following PowerShell scripts to set up SnapMirror and perform failover.

2.2.1 POWERSHELL SCRIPT: SET UP SNAPMIRROR --

```
# Import the ONTAP PowerShell Toolkit cmdlets
Import-Module DataONTAP

# Connect to the filer to set the user credentials, prompts user for password
Connect-NaController FAS3140-MPSC-2 -credential admin

# Restrict the NetApp destination volume for the initial SnapMirror transfer
Set-NaVol SMHV_VMData_Mirror -Restricted

# Initialize SnapMirror to start replicating the data to the failover site
Invoke-NaSnapmirrorInitialize -source FAS3140-MPSC-1:SMHV_VMData -destination FAS3140-
    MPSC-2:SMHV_VMData_Mirror

# Show the status of the SnapMirror replication
Get-NaSnapmirror |ft -property SourceLocation, DestinationLocation, Status | Out-Default

Write-Host "Use Get-NaSnapmirror command to view the SnapMirror status of the Volumes.
    Perform further actions only after the Status is In-Sync"

-- End Setup –
```

2.2.2 VBScript: Import Virtual Machine Configuration --

```
option explicit

dim objWMIService
dim managementService
dim switchService
```

```
dim fileSystem
const JobStarting = 3
const JobRunning = 4
const JobCompleted = 7
const wmiStarted = 4096
const wmiSuccessful = 0


Main()
'---------------------------------------------------------------------
' Main
'---------------------------------------------------------------------
Sub Main()
	dim computer, objArgs, importDirectory, generateNewID,VHDName, VMName, Suppress
	set fileSystem = Wscript.CreateObject("Scripting.FileSystemObject")
	computer = "."
	set objWMIService = GetObject("winmgmts:\\" & computer & "\root\virtualization")
	set managementService = objWMIService.ExecQuery("select * from
Msvm_VirtualSystemManagementService").ItemIndex(0)
	set switchService = objWMIService.ExecQuery("select * from
Msvm_VirtualSwitchManagementService").ItemIndex(0)
	set objArgs = WScript.Arguments
	if WScript.Arguments.Count = 3 then
		importDirectory = objArgs.Unnamed.Item(0)
		VHDName = objArgs.Unnamed.Item(1)
		VMName = objArgs.Unnamed.item(2)
	else
		WScript.Echo "usage: cscript ImportVirtualSystemEx-ConfigOnly.vbs
importDirectoryName VHDNAME.vhd VMName"
		WScript.Quit(1)
	end if
	if (ImportVirtualSystemEx(importDirectory,VHDName,VMName)) then
		WScript.Quit(0)
	end if
		wscript.quit(0)
End Sub


'---------------------------------------------------------------------
' GetVirtualSystemImportSettingData from a directory
'---------------------------------------------------------------------
Function GetVirtualSystemImportSettingData(importDirectory)
	dim objInParam, objOutParams
	set objInParam =
managementService.Methods_("GetVirtualSystemImportSettingData").InParameters.SpawnInstanc
e_()
	objInParam.ImportDirectory = importDirectory
```

```
	set objOutParams =
managementService.ExecMethod_("GetVirtualSystemImportSettingData", objInParam)
	if objOutParams.ReturnValue = wmiStarted then
		if (WMIJobCompleted(objOutParams)) then
			set GetVirtualSystemImportSettingData =
objOutParams.ImportSettingData
		end if
	elseif objOutParams.ReturnValue = wmiSuccessful then
		set GetVirtualSystemImportSettingData = objOutParams.ImportSettingData
	else
		WriteLog Format1("GetVirtualSystemImportSettingData failed with
ReturnValue {0}", objOutParams.ReturnValue)
	end if
End Function


'-----------------------------------------------------------------
' ImportVirtualSystem from a directory
'-----------------------------------------------------------------
Function ImportVirtualSystemEx(importDirectory,VHDName, VMName)


	dim objInParam, objOutParams
	dim newDataRoot
	dim importSettingData
	dim newSourceResourcePaths, newTargetNetworkConnections, newSwitch


	'Resources in newSourceResourcePaths below should be existing. Fill this with the
resources corresponding to those in CurrentResourcePaths
	newSourceResourcePaths = Array(1)
	newSourceResourcePaths(0) = importDirectory & "\" & VHDName
	ImportVirtualSystemEx = false
	set objInParam =
managementService.Methods_("ImportVirtualSystemEx").InParameters.SpawnInstance_()
	objInParam.ImportDirectory = importDirectory
	set importSettingData = GetVirtualSystemImportSettingData(importDirectory)
	importSettingData.GenerateNewId = true
	importSettingData.CreateCopy = false
	importSettingData.Name = VMName
	importSettingData.SourceResourcePaths = newSourceResourcePaths
	importSettingData.Put_
	objInParam.ImportSettingData = importSettingData.GetText_(1)
	set objOutParams = managementService.ExecMethod_("ImportVirtualSystemEx",
objInParam)


	if objOutParams.ReturnValue = wmiStarted then
		if (WMIJobCompleted(objOutParams)) then
```

```
                        ImportVirtualSystemEx = true
                end if
        elseif objOutParams.ReturnValue = wmiSuccessful then
                ImportVirtualSystemEx = true
        end if
End Function

'---------------------------------------------------------------------
' Handle wmi Job object
'---------------------------------------------------------------------
Function WMIJobCompleted(outParam)
        dim WMIJob, jobState
        set WMIJob = objWMIService.Get(outParam.Job)
        WMIJobCompleted = true
        jobState = WMIJob.JobState

        while jobState = JobRunning or jobState = JobStarting
                WScript.Sleep(1000)
                set WMIJob = objWMIService.Get(outParam.Job)
                jobState = WMIJob.JobState
        wend

End Function

'---------------------------------------------------------------------
' Create the console log files.
'---------------------------------------------------------------------
Sub WriteLog(line)
        dim fileStream
        set fileStream = fileSystem.OpenTextFile(".\ImportVirtualSystemEx-ConfigOnly.log",
8, true)
        WScript.Echo line
        fileStream.WriteLine line
        fileStream.Close
End Sub

'---------------------------------------------------------------------------
' The string formating functions to avoid string concatenation.
'---------------------------------------------------------------------------
Function Format1(myString, arg0)
    Format1 = Replace(myString, "{0}", arg0)
End Function
```

2.2.3 VBSCRIPT: EXPORT VIRTUAL MACHINE CONFIGURATION –

```
option explicit

dim objWMIService
dim managementService
dim fileSystem

const JobStarting = 3
const JobRunning = 4
const JobCompleted = 7
const wmiStarted = 4096
const wmiSuccessful = 0

Main()
'---------------------------------------------------------------------
' Main
'---------------------------------------------------------------------
Sub Main()
     dim computer, objArgs, vmName, vm, exportDirectory
        set fileSystem = Wscript.CreateObject("Scripting.FileSystemObject")
        computer = "."
        set objWMIService = GetObject("winmgmts:\\" & computer & "\root\virtualization")
        set managementService = objWMIService.ExecQuery("select * from
Msvm_VirtualSystemManagementService").ItemIndex(0)
        set objArgs = WScript.Arguments
        if WScript.Arguments.Count = 2 then
               vmName = objArgs.Unnamed.Item(0)
               exportDirectory = objArgs.Unnamed.Item(1)
        else
               WScript.Echo "usage: cscript ExportVirtualSystemEx.vbs vmName
exportDirectoryName"
               WScript.Quit(1)
        end if
        set vm = GetComputerSystem(vmName)
        if ExportVirtualSystemEx(vm, exportDirectory) then
               WriteLog "Done"
               WScript.Quit(0)
       else
               WriteLog "ExportVirtualSystemEx Failed."
               WScript.Quit(1)
        end if
End Sub
'---------------------------------------------------------------------
```

```
' Retrieve Msvm_VirtualComputerSystem from base on its ElementName
'-----------------------------------------------------------------
Function GetComputerSystem(vmElementName)
    On Error Resume Next
    dim query
    query = Format1("select * from Msvm_ComputerSystem where ElementName = '{0}'",
vmElementName)
    set GetComputerSystem = objWMIService.ExecQuery(query).ItemIndex(0)
    if (Err.Number <> 0) then
               'WriteLog Format1("Err.Number: {0}", Err.Number)
        'WriteLog Format1("Err.Description:{0}",Err.Description)
              WScript.Quit(1)
    end if
End Function
'-----------------------------------------------------------------
' Export a virtual system
'-----------------------------------------------------------------
Function ExportVirtualSystemEx(computerSystem, exportDirectory)
    dim objInParam, objOutParams
        dim query
        dim computer
        dim exportSettingData
        ExportVirtualSystemEx = false
        if Not fileSystem.FolderExists(exportDirectory) then
               fileSystem.CreateFolder(exportDirectory)
    end if
        set objInParam =
managementService.Methods_("ExportVirtualSystemEx").InParameters.SpawnInstance_()
        objInParam.ComputerSystem = computerSystem.Path_.Path
        query = Format1("ASSOCIATORS OF {{0}} WHERE resultClass =
Msvm_VirtualSystemExportSettingData", computerSystem.Path_.Path)
    set exportSettingData = objWMIService.ExecQuery(query).ItemIndex(0)
        'Dont copy the VHDs and AVHDs, but copy the Saved state information and Snapshot
configurations if present
        exportSettingData.CopyVmStorage = false
        exportSettingData.CopyVmRuntimeInformation = true
        exportSettingData.CreateVmExportSubdirectory = true
        exportSettingData.CopySnapshotConfiguration = 0
        objInParam.ExportSettingData = exportSettingData.GetText_(1)
        objInParam.ExportDirectory = exportDirectory
        set objOutParams = managementService.ExecMethod_("ExportVirtualSystemEx",
objInParam)
        if objOutParams.ReturnValue = wmiStarted then
               if (WMIJobCompleted(objOutParams)) then
                      ExportVirtualSystemEx = true
```

```
                end if
        elseif (objOutParams.ReturnValue = wmiSuccessful) then
                ExportVirtualSystemEx = true
        else
                WriteLog Format1("ExportVirtualSystemEx failed with ReturnValue {0}",
objOutParams.ReturnValue)
    end if
End Function
'---------------------------------------------------------------------
' Handle wmi Job object
'---------------------------------------------------------------------
Function WMIJobCompleted(outParam)
        dim WMIJob, jobState
        set WMIJob = objWMIService.Get(outParam.Job)
        WMIJobCompleted = true
        jobState = WMIJob.JobState
        while jobState = JobRunning or jobState = JobStarting
                WriteLog Format1("In progress... {0}% completed.",WMIJob.PercentComplete)
                WScript.Sleep(1000)
                set WMIJob = objWMIService.Get(outParam.Job)
                jobState = WMIJob.JobState
    wend
        if (jobState <> JobCompleted) then
                WriteLog Format1("ErrorCode:{0}", WMIJob.ErrorCode)
                WriteLog Format1("ErrorDescription:{0}", WMIJob.ErrorDescription)
                WMIJobCompleted = false
        end if
End Function
'---------------------------------------------------------------------
' Create the console log files.
'---------------------------------------------------------------------
Sub WriteLog(line)
    dim fileStream
    'set fileStream = fileSystem.OpenTextFile(".\ExportVirtualSystemExLog.log", 8, true)
    WScript.Echo line
    'fileStream.WriteLine line
    'fileStream.Close
End Sub
'---------------------------------------------------------------------------
' The string formating functions to avoid string concatenation.
'---------------------------------------------------------------------------
Function Format2(myString, arg0, arg1)
    Format2 = Format1(myString, arg0)
```

```
    Format2 = Replace(Format2, "{1}", arg1)
End Function
'-------------------------------------------------------------------------
' The string formating functions to avoid string concatenation.
'-------------------------------------------------------------------------
Function Format1(myString, arg0)
    Format1 = Replace(myString, "{0}", arg0)
End Function
```

2.2.4 POWERSHELL SCRIPT: FAILOVER VIRTUAL MACHINES TO THE DR SITE –

```
#-----------------------------------------------------
# -- Steps in following Script                                              --
# -- 1). remove mapping from Main servers to Main Array --
# -- 2). Stop and delete MAIN Site VMs. Shutdown CSV --
# -- 3). Flip Snapmirror relationship. DR is LIVE          --
# -- 4). Map CSV to DR Servers,Activate DR Array           --
# -- 5). Copy Import Files into place, and import VM --
# -- 6). Bring VMs into Cluster on DR Site                 --
# -----------------------------------------------------


# -----------------------------------------------
# -- Begin Edit Block                           --
# -- Only Edit Variables in the following Section.  --
# -----------------------------------------------
$VMList                             = @("SQL10",
"WEB10","EXCH10","WUS10","Sharepoint10","DC10")
$FQDN2                              = "sear-host2"
$FQDN3                              = "sear-host3"
$FQDN4                              = "sear-host4"
$FQDN5                              = "taco-host2"
$FQDN6                              = "taco-host3"
$MainFiler                  = "seareal3070a"
$DRFiler                            = "seareal3070b"
$MainLUNName                = "/vol/CSV3/CSV3PRI"
$DRLUNName                          = "/vol/CSV3/CSV3PRI"
$admin                      = "snapdrive"
$MainSiteServer                     = $FQDN2
$DRSiteServer               = $FQDN6
$CSVVolumeName              = "Cluster Disk 3"
$MainCSVVolName                     = "Cluster Disk 4"
$DRCSVName                          = "Cluster Disk 3"
#--------
# These are Autogenerated. You dont need to edit these.
```

```
#-----------
$temp = get-cluster -name $MainSiteServer
$MainCluster = $temp.name
$temp = get-cluster -name $DRSiteServer
$DRCluster = $temp.name
$iqn2 = "iqn.1991-05.com.microsoft:" + $FQDN2 + ".seattle.retail"
$iqn3 = "iqn.1991-05.com.microsoft:" + $FQDN3 + ".seattle.retail"
$iqn4 = "iqn.1991-05.com.microsoft:" + $FQDN4 + ".seattle.retail"
$iqn5 = "iqn.1991-05.com.microsoft:" + $FQDN5 + ".seattle.retail"
$iqn6 = "iqn.1991-05.com.microsoft:" + $FQDN6 + ".seattle.retail"
$igroup2 = "viaRPC." + $iqn2
$igroup3 = "viaRPC." + $iqn3
$igroup4 = "viaRPC." + $iqn4
$igroup5 = "viaRPC." + $iqn5
$igroup6 = "viaRPC." + $iqn6
$password = ConvertTo-SecureString "net@pp1" -AsPlainText -Force
$cred = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
$admin,$password
$udpatesnapdr = "sealeal3070b:CSV3"
$updatesnapmain = "sealeal3070a:CSV3"
$ConfirmPreference = "None"

function WaitTimer {param ([int]$Counter1)
        Write-host -foregroundcolor green "$Counter1.." -nonewline
        if ($Counter1 -gt 1){
                start-sleep -s 1
                $Counter1=$Counter1-1
                WaitTimer ($Counter1)}
        if ($Counter1 -eq 1) {Write-host -foregroundcolor green ".Seconds"}
        }
cls
echo "------------------------------------------------"
echo "- Starting Failover from Main Site to DR Site    -"
echo "------------------------------------------------"
Connect-naController $MainFiler -credential $cred
Connect-naController $DRFiler -credential $cred

foreach ($VM in $VMList)
{       Echo "- ---Stoping VM on Main Site : $VM"
        stop-vm $VM -server $MAINSiteServer -ea silentlycontinue -force
}
WaitTimer 6
foreach ($VM in $VMList)
```

```
{       Echo "- ---Removing Cluster Resources attached to VM on Main Site : $VM"
        Remove-ClusterGroup $VM -RemoveResources -Cluster $MAINCluster -ea
silentlycontinue -force
}
WaitTimer 6
foreach ($VM in $VMList)
{       Echo "- ---Removing Actual VM from Hyper V on Main Site : $VM"
        remove-vm $VM -server $MAINSiteServer -ea silentlycontinue -force
}
WaitTimer 6
Echo "- ---Removing Cluster Resource on Primary Site"
Stop-ClusterResource $MainCSVVolName -Cluster $MainCluster -ea silentlycontinue
Echo "- ---Removing Mapping from Primary Servers to Primary Array"
remove-nalunmap $MainLUNName $igroup2 -controller $MainFiler -ea silentlycontinue
remove-nalunmap $MainLUNName $igroup3 -controller $MainFiler -ea silentlycontinue
remove-nalunmap $MainLUNName $igroup4 -controller $MainFiler -ea silentlycontinue
Echo "- ---Updating Mirror Relationship"
invoke-NaSnapMirrorUpdate  -source $updatesnapdr -destination $updatesnapmain -controller
$DRFiler
WaitTimer 3
Echo "- ---Reverse Mirror Relationship"
Invoke-NaSnapMirrorResync -Source seareal3070b:CSV3 seareal3070a:CSV3
WaitTimer 3


Echo "- ---Mapping LUN from Secondary Array to Secondary Cluster"
Add-NaLunMap -path $DRLUNName -initiatorgroup $igroup5 -controller $DRFiler -ea
silentlycontinue -force
Add-NaLunMap -path $DRLUNName -initiatorgroup $igroup6 -controller $DRFiler -ea
silentlycontinue -force


WaitTimer 10
"rescan" | Diskpart


Echo "- ---Enabling Cluster Shared Volume on Secondary Site"
Start-ClusterResource $CSVVolumeName -Cluster $DRCluster
$CSVCheck=get-clustersharedvolume $CSVVolumeName
$timeout=30
do {Start-Sleep -s 2
        Echo "Waiting for $CSVVolumeName to come online"
        if ($CSVCheck.state -ne "Failed")
                {$timeout = 0}
        $Timeout=$Timeout-2
} while ($Timeout -gt 0)
if ($CSVCheck.state -eq "Failed")
```

```
        {       Echo "CSV Never showed up on remote site"
                exit(1) }
Echo "- ---Updating the Reversed Mirror. Secondary Site and Primary Site roles now
reversed"
Connect-naController $MainFiler -credential $cred
Invoke-NaSnapMirrorResync -Source seareal3070b:CSV3 seareal3070a:CSV3
WaitTimer 6
foreach ($VM in $VMList)
{       Echo "- ---Copy VM Import Files into VM Directories. Note : NOT Moving VHDs."
        xcopy C:\ClusterStorage\Volume2\Exports\$VM C:\ClusterStorage\Volume2\$VM /s /y
        $VMVHD=$VM+".vhd"
        Echo "- ---Import The VM via a Config Only VM Import. Specify existing in place
VHD"
        if (Get-VM $VM) {write-host "- ---VM Named $VM already Exists. Skipping Import"}
        else {C:\Software\ConfigOnlyImportv3 C:\ClusterStorage\Volume2\$VM $VMVHD $VM
                  write-host $VM
                  waittimer 5}
}
WaitTimer 6
foreach ($VM in $VMList)
{       Echo "- ---Add newly imported VM to Cluster"
        If ((get-cluster "Virtual Machine $VM" -ea silentlycontinue)) {
                write-host "- ---Cluster Resource Group Named $VM already Exists. Removing
Old Clus Resource Group"
                Remove-ClusterGroup $VM -RemoveResources -Cluster $DRCluster-force
                waittimer 3}

        Add-ClusterVirtualMachineRole -VirtualMachine $VM -Cluster $DRCluster
}
WaitTimer 6
foreach ($VM in $VMList)
{       Write-host "- ---Start VM" -nonewline
        Start-VM $VM -ea silentlycontinue
}
Echo "Failover to DR Site Complete"
```

2.2.5 POWERSHELL SCRIPT: FAILOVER VIRTUAL MACHINES BACK TO THE MAIN SITE –

```
#-----------------------------------------------------
# -- Steps in following Script                                              --
# -- 1). remove mapping from DR servers to DR Array --
# -- 2). Stop and delete DR Site VMs. Shutdown CSV  --
# -- 3). Flip Snapmirror relationship. MAIN is LIVE --
# -- 4). Map CSV to Main Servers,Activate MAIN Array--
# -- 5). Copy Import Files into place, and import VM--
```

```
# -- 6). Bring VMs into Cluster on Main Site        --
# ---------------------------------------------------

# ---------------------------------------------------
# -- Begin Edit Block --
# -- Only Edit Variables in the following Section.  --
# ---------------------------------------------------
$VMList = @("SQL10", "WEB10","EXCH10","WUS10","Sharepoint10","DC10")
$FQDN2 = "sear-host2"
$FQDN3  = "sear-host3"
$FQDN4 = "sear-host4"
$FQDN5  = "taco-host2"
$FQDN6  = "taco-host3"
$MainSiteServer = "10.58.96.228"
$DRSiteServer  = "10.58.96.225"
$MainFiler = "seareal3070a"
$DRFiler = "seareal3070b"
$MainLUNName = "/vol/CSV3/CSV3PRI"
$DRLUNName = "/vol/CSV3/CSV3PRI"
$admin = "administrator"
$udpatesnapdr  = "seareal3070b:CSV3"
$updatesnapmain = "sealeal3070a:CSV3"
$CSVVolumeName = "Cluster Disk 4"
$CSVVolumePath = "C:\ClusterStorage\Volume3\"
#--------
# These are Autogenerated. You dont need to edit these.
#-----------
$temp = get-cluster -name $MainSiteServer
$MainCluster = $temp.name
$temp = get-cluster -name $DRSiteServer
$DRCluster = $temp.name
$iqn2 = "iqn.1991-05.com.microsoft:" + $FQDN2 + ".seattle.retail"
$iqn3 = "iqn.1991-05.com.microsoft:" + $FQDN3 + ".seattle.retail"
$iqn4 = "iqn.1991-05.com.microsoft:" + $FQDN4 + ".seattle.retail"
$iqn5 = "iqn.1991-05.com.microsoft:" + $FQDN5 + ".seattle.retail"
$iqn6 = "iqn.1991-05.com.microsoft:" + $FQDN6 + ".seattle.retail"
$igroup2 = "viaRPC." + $iqn2
$igroup3 = "viaRPC." + $iqn3
$igroup4 = "viaRPC." + $iqn4
$igroup5 = "viaRPC." + $iqn5
$igroup6 = "viaRPC." + $iqn6
$password = ConvertTo-SecureString "net@pp1" -AsPlainText -Force
```

```
$cred  = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList
$admin,$password
$ConfirmPreference = "None"


function WaitTimer {param ([int]$Counter1)
       Write-host -foregroundcolor green ". ...$Counter1" -nonewline
       if ($Counter1 -gt 1){
              start-sleep -s 1
              $Counter1=$Counter1-1
              WaitTimer ($Counter1)}
       if ($Counter1 -le 1) {Write-host -foregroundcolor green ".Seconds"}
       }
cls
echo "------------------------------------------------"
echo "- Starting Failback from DR Site to Main Site    -"
echo "------------------------------------------------"
Connect-naController $MainFiler -credential $cred
Connect-naController $DRFiler -credential $cred

foreach ($VM in $VMList)
{      Echo "----Stoping VM on DR Site : $VM"
       stop-vm $VM -server $DRSiteServer -ea silentlycontinue -force
}
WaitTimer 5
foreach ($VM in $VMList)
{      Echo "----Removing Cluster Resource Groups for VM : $VM"
       Remove-ClusterGroup $VM -RemoveResources -Cluster $DRCluster -ea silentlycontinue
-force
}
WaitTimer 5
foreach ($VM in $VMList)
{      Echo "----Removing actual VM from Hyper V : $VM"
       remove-vm $VM -server $DRSiteServer -ea silentlycontinue -force
}
Echo "- ---Removing Cluster Resource on Primary Site"
Stop-ClusterResource "Cluster Disk 3" -Cluster $DRCluster -ea silentlycontinue

Echo "- ---Removing Mapping from Primary Servers to Primary Array"
remove-nalunmap $DRLUNName $igroup5 -controller $DRFiler -ea silentlycontinue
remove-nalunmap $DRLUNName $igroup6 -controller $DRFiler -ea silentlycontinue

Echo "- ---Updating Mirror Relationship"
invoke-NaSnapMirrorUpdate  -source $updatesnapdr -destination $updatesnapmain -controller
$DRFiler
```

```
WaitTimer 3

Echo "- ---Reverse Mirror Relationship"

Invoke-NaSnapMirrorResync -Source seareal3070b:CSV3 seareal3070a:CSV3

WaitTimer 3


Echo "- ---Mapping LUN from Secondary Array to Secondary Cluster"

Add-NaLunMap -path $MainLUNName -initiatorgroup $igroup2 -controller $MainFiler -force -
ea silentlycontinue

Add-NaLunMap -path $MainLUNName -initiatorgroup $igroup3 -controller $MainFiler -force -
ea silentlycontinue

Add-NaLunMap -path $MainLUNName -initiatorgroup $igroup4 -controller $MainFiler -force -
ea silentlycontinue

WaitTimer 10

"rescan" | Diskpart


Echo "- ---Enabling Cluster Shared Volume on Main Site"

Start-ClusterResource $CSVVolumeName -Cluster $MainCluster

$CSVCheck=get-clustersharedvolume $CSVVolumeName

$timeout=30

do {Start-Sleep -s 2
       Write-host "Waiting for $CSVVolumeName to come online"

       $CSVCheck=get-clustersharedvolume $CSVVolumeName

       if ($CSVCheck.state -ne "Failed")

              {$timeout = 0}

       $Timeout=$Timeout-2

} while ($Timeout -gt 0)

if ($CSVCheck.state -eq "Failed")

       {      Echo "CSV Never showed up on remote site"

              exit(1) }

Echo "- ---Updating the Reversed Mirror. Secondary Site and Primary Site roles now
reversed"

Connect-naController $MainFiler -credential $cred

Invoke-NaSnapMirrorResync -Source seareal3070b:CSV3 seareal3070a:CSV3

WaitTimer 10

Echo "- ---Copy VM Import Files into VM Directories and Import them via Config Only
Import."

foreach ($VM in $VMList)

       {      Echo "- ---Copy VM Import Files into VM Directories. Note : NOT Moving
VHDs."

              $VMFullpath = $CSVVolumePath+"\"+$VM

              $VMExportPath = $CSVVolumePath+"\exports\"+$VM

              xcopy $VMExportPath $VMFullPath /s /y

       }

foreach ($VM in $VMList)

       {      $VMVHD=$VM+".vhd"
```

```
                $VMFullpath = $CSVVolumePath+"\"+$VM
                Echo "- ---Import The VM via a Config Only VM Import. Specify existing in
place VHD"
                if (Get-VM $VM) {write-host "- ---VM Named $VM already Exists. Skipping
Import"}
                else {
                        C:\Software\ConfigOnlyImportv3 $VMFullPath $VMVHD $VM
                        write-host $VM
                        waittimer 5}
        }
WaitTimer 6
foreach ($VM in $VMList)
        {       Echo "- ---Add newly imported VM to Cluster"
                If (get-cluster "Virtual Machine $VM" -ea silentlycontinue) {
                        write-host "- ---Cluster Resource Group Named $VM already Exists.
Removing Old Clus Resource Group"
                        Remove-ClusterGroup $VM -RemoveResources -Cluster $MainCluster -ea
silentlycontinue -force
                        waittimer 3}
                Add-ClusterVirtualMachineRole -VirtualMachine $VM -Cluster $MainCluster
        }
WaitTimer 3
foreach ($VM in $VMList)
        {       Echo "- ---Start VM"
                Start-VM $VM
        }


Echo "Failback to Main Site Complete"
```

2.3 SNAPMANAGER FOR HYPER-V

The scripts in section 2.2 facilitate virtual machine recovery in a crash-consistent state to the standby site.

NetApp SnapManager for Hyper-V (SMHV) leverages the virtual machine backup and restore by using the underlying Snapshot technology. It offloads the backup activity to storage, thereby reducing the load on the CPU and the network. It integrates with Microsoft Hyper-V VSS writer to quiesce a virtual machine (VM) before creating an application-consistent Snapshot copy of the VM. SMHV is a VSS Requestor and coordinates the backup operation to create a consistent Snapshot copy, using the VSS Hardware Provider for NetApp Data ONTAP®. It allows you to create application-consistent backups of a VM if you have Microsoft Exchange, Microsoft SQL Server®, or any other VSS-aware application running on VHDs in the VM. The applications that exist in the VM are restored to the same state they were in at the time of the backup. SMHV restores the VM to its original location quickly and efficiently, thereby reducing administrative overhead

TR-3805 (http://media.netapp.com/documents/tr-3805.pdf) offers a detailed description of the features and best practices for using the SnapMirror for Hyper-V application.

3 CONCLUSION

By using NetApp disaster recovery capabilities for the Microsoft Dynamic Data Center, based on Windows Server 2008 R2 Hyper-V, you can:

- Expand your business by addressing critical customer needs with new disaster recovery services
- Get to revenue faster by leveraging NetApp expertise with Microsoft environments
- Minimize risk with reliable disaster recovery services
- Reduce costs with efficient storage infrastructure and streamlined IT administration

For additional supporting information about the joint disaster recovery capabilities, go to the Dynamic Data Center Toolkit Web site:
http://www.microsoft.com/hosting/dynamicdatacenter/Home.html

To provide feedback and to request further support from NetApp as you test these capabilities, go to the NetApp Communities Web site:
http://communities.netapp.com/community/interfaces_and_tools.

To discuss a comprehensive strategy for how NetApp can help you build a successful service business, please contact your NetApp sales representative.