Guide Opérationnel :
Metro Availability avec VMware vSphere

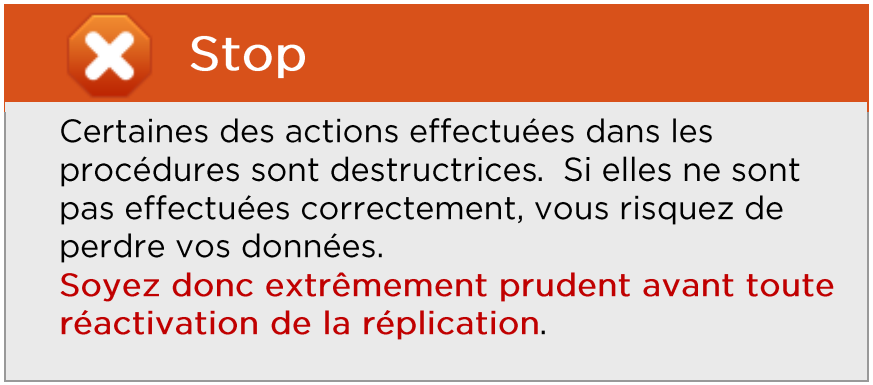
Table des Matières

Synopsis

Ce document est un guide opérationnel décrivant comment utiliser la fonctionnalité de réplication synchrone sur Nutanix, appelée Metro Availability, en combinaison avec l'hyperviseur vSphere. L'audience ciblée est l'équipe opérationnelle de production qui gère la plateforme Nutanix et qui devra réaliser les opérations de bascules, planifiées ou non.

Ce guide détaille les deux opérations suivantes :

1. Comment effectuer une bascule planifiée d'un « protection domain » (PD) de type « Metro Availability » (MA) : cette procédure est à utiliser lorsqu'un des deux sites du géo-cluster doit être arrêté pour une opération de maintenance et que vous souhaitez migrer l'ensemble des machines virtuelles ainsi que leur stockage sur le second site (opération qui peut s'effectuer sans interruption de service depuis la version 4.6 de l'Acropolis Operating System (AOS)). La procédure explique également comment effectuer le retour arrière.
2. Que faire en cas de désastre : cette procédure explique comment récupérer l'ensemble des machines virtuelles ainsi que leur stockage quand un sinistre est intervenu sur l'un des deux sites. La procédure de retour arrière est également expliquée.



Procédure #1 : Bascule planifiée

Introduction

Vous effectuez une bascule planifiée quand vous avez besoin de migrer l'ensemble de vos machines virtuelles (VMs) d'un site source vers un site cible afin de pouvoir arrêter complètement le site source pour une opération de maintenance. Une bascule planifiée peut également avoir lieu lors d'un exercice de plan de reprise d'activité (PRA).

Une fois le PRA ou l'opération de maintenance sur le site primaire terminé, vous migrerez ensuite les machines virtuelles concernées à nouveau vers le site source pour faire retour arrière.

Cette procédure utilise la dénomination de site source pour désigner le site qui contient les machines virtuelles et le stockage à migrer. Le site cible est le site vers lequel ces machines virtuelles et ce stockage seront migrés.

Dans les exemples utilisés, le site source s'appelle dc1, et le site cible s'appelle dc2.

Pour le retour arrière, la migration s'effectue du site cible (dc2) vers le site source (dc1).

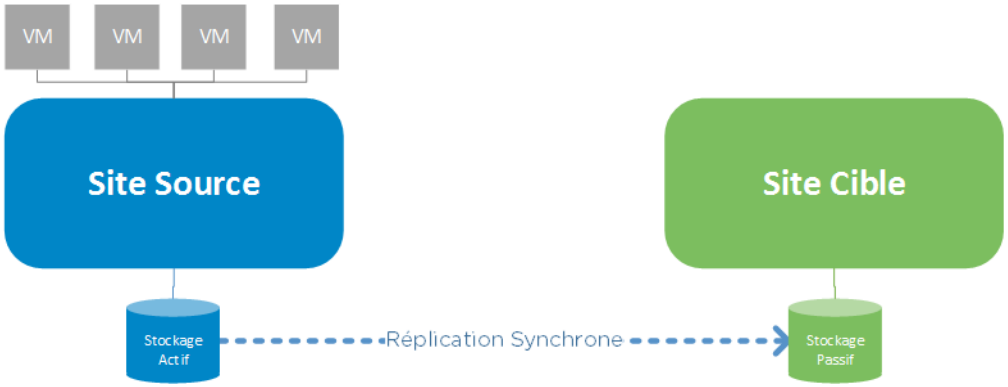
Chaque procédure est décrite en détail mais contient aussi un schéma de :

1. Situation initiale : ce schéma montre la situation de départ. Où sont les VMs, où le stockage est actif et dans quel sens s'effectue la réplication.
2. Un flux de travail : il décrit les grandes étapes de la procédure. Les actions destructrices sont indiquées en orange et labélisées avec un point d'exclamation dans un triangle. Avant d'effectuer ces actions, assurez-vous que vos machines virtuelles sont sur le bon site, sinon vous risquez de perdre des données et une interruption de service.

3. Un arrêté de situation : ce schéma montre où sont les VMs, le stockage actif et le sens de la réplication une fois que la procédure a été déroulée.

Bascule de la source vers la cible

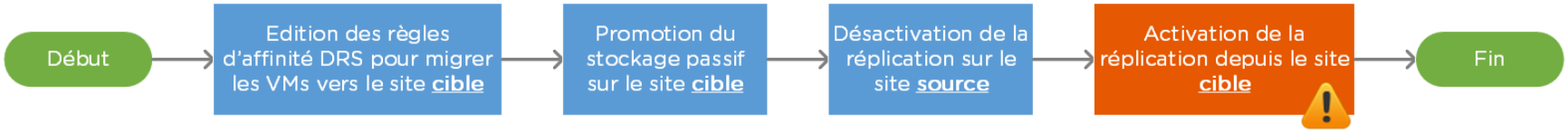
Situation initiale :



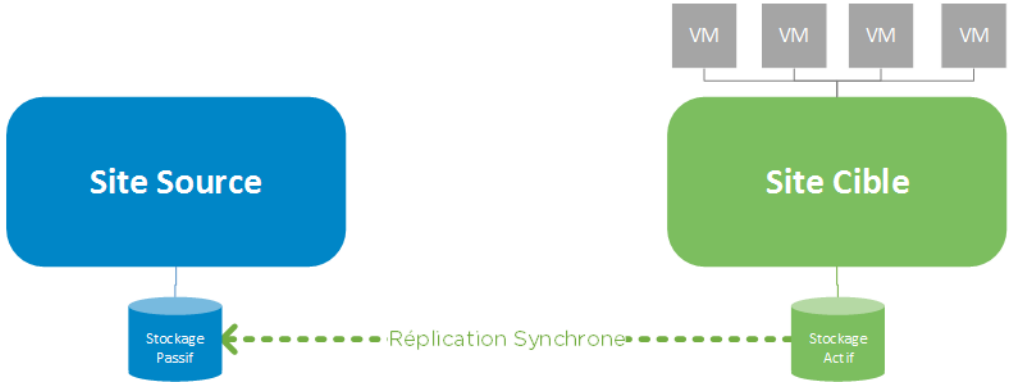
Notons que :

1. Les VMs tournent sur le site source, dc1, où le stockage est actif.
2. La réplication synchrone s'effectue du site source (dc1) vers le site cible (dc2)

Le flux de travail est le suivant :



L'arrêté de situation une fois la procédure déroulée sera le suivant :



Notons que :

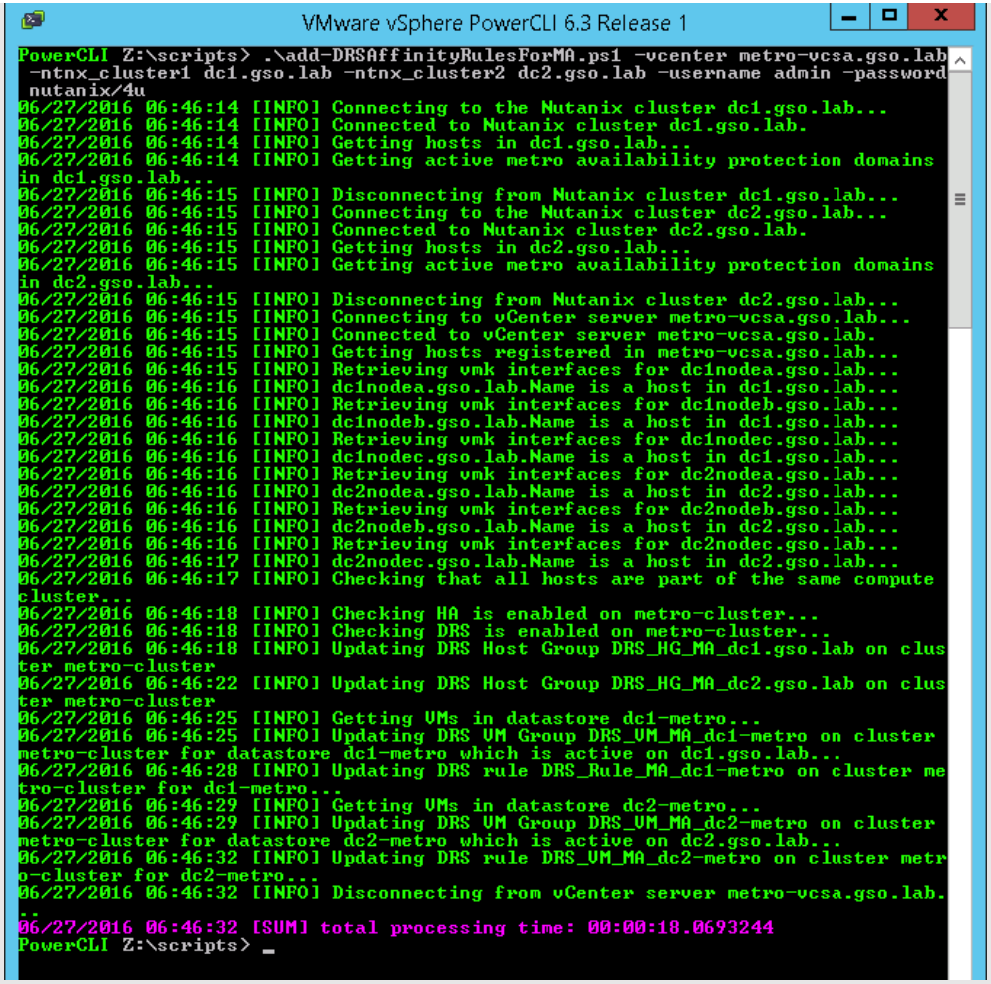
1. Les VMs tourneront sur le site cible (dc2)
2. Le stockage sera actif sur le site cible (dc2)
3. La réplication synchrone s'effectuera du site cible (dc2) vers le site source (dc1)

Pour le détail de la procédure ci-dessous, notons que :

1. Le site source est *dc1*
2. Le site cible est *dc2*
3. Le « container » (entité logique de stockage) est *dc1-metro*
4. Le « protection domain » (entité logique de réplication) est également appelé *dc1-metro*
5. Les VMs *vm1* à *vm10* tournent sur *dc1*

6. Le cluster vSphere est *metro-cluster*
7. Certaines captures d'écran montrent d'autres « containers » et « protection domains » car dc2 est également répliqué vers dc1, comme c'est le cas dans la plupart des environnements de production.

1.



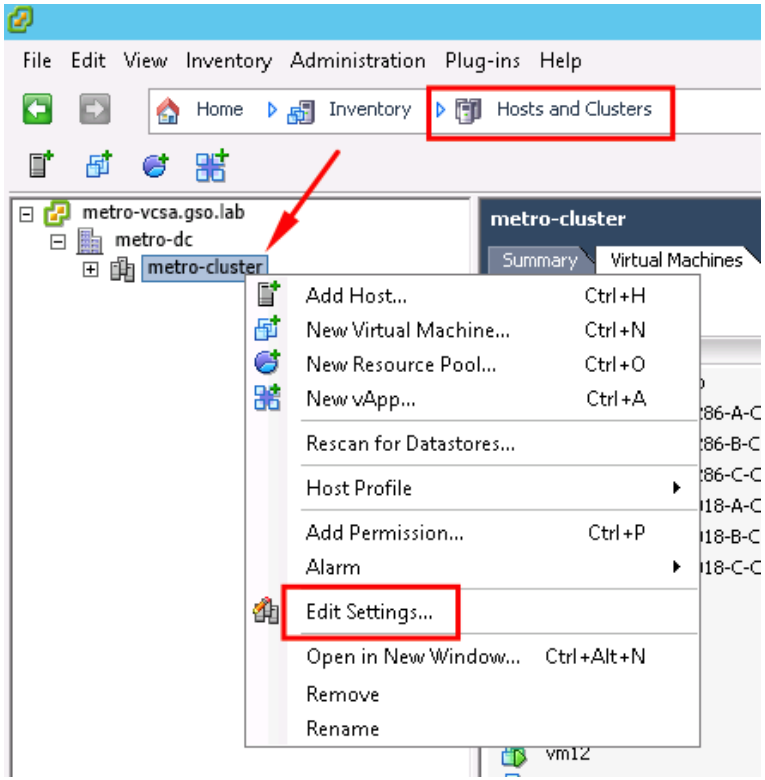
La première étape consiste à s'assurer que les règles d'affinité et les groupes DRS sont à jour.
Cette étape est cruciale car si DRS n'est pas à jour, lorsque nous désactiverons la réplication, certaines VMs ne seront pas sur le bon site. Lorsque la réplication sera réactivée, ces VMs verrons leurs disques écrasés.

Afin de mettre à jour les règles DRS, nous utilisons le script PowerShell **add-DRSAffinityRulesForMA.ps1** qui est disponible dans l'appendice de ce document.

Note

Le script créera les groupes et règles DRS s'ils n'existent pas, ou les mettra à jour si ils existent déjà.

2.



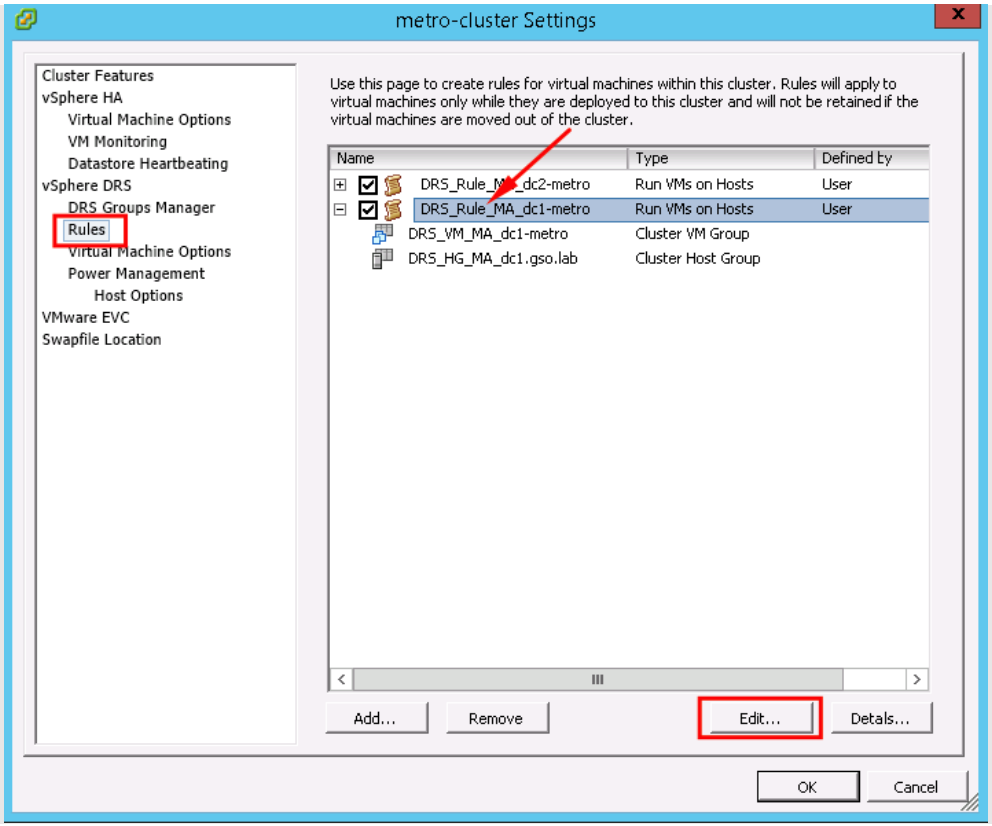
Stop

Avant de continuer, assurez-vous dans Prism que le « protection domain » *dc1-metro* est bien actif (active) sur *dc1* et passif (standby) sur *dc2* et que le statut est « **Enabled (in sync)** »

Maintenant que nos règles d'affinité DRS sont à jour, nous éditons la règle DRS *dc1-metro* afin que les machines virtuelles dans cet espace de stockage soient migrées vers des hôtes de *dc2*.

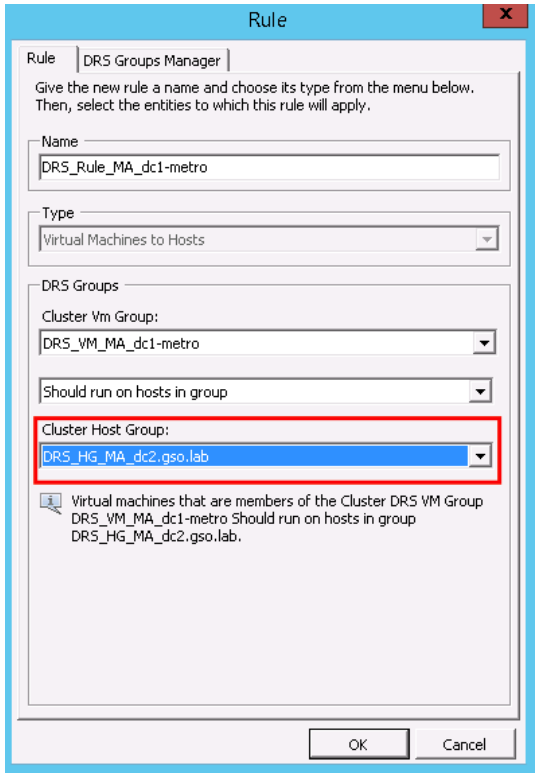
Pour ce faire, commencez par faire un clic droit sur le cluster dans le client vSphere et sélectionnez « **Edit Settings** »

3.

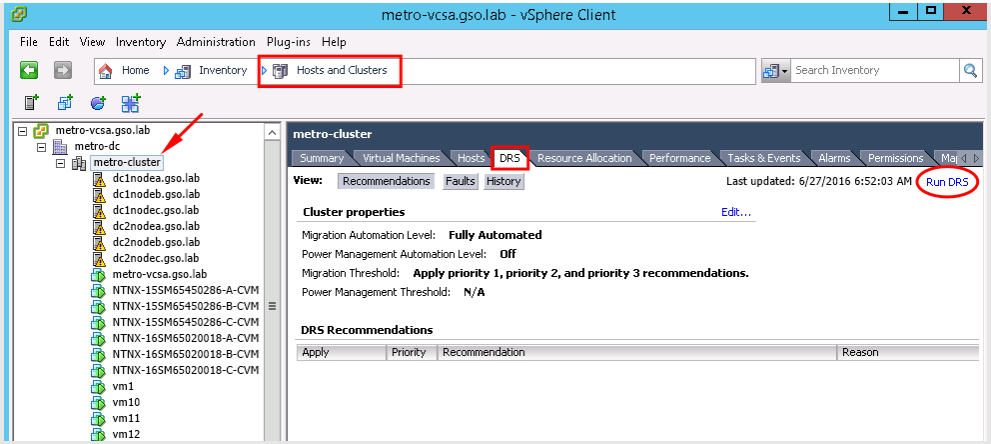


Sélectionnez « **Rules** » sur le panneau de gauche, puis sélectionnez la règle pour *dc1-metro* et cliquez sur « **Edit** » en bas à droite.

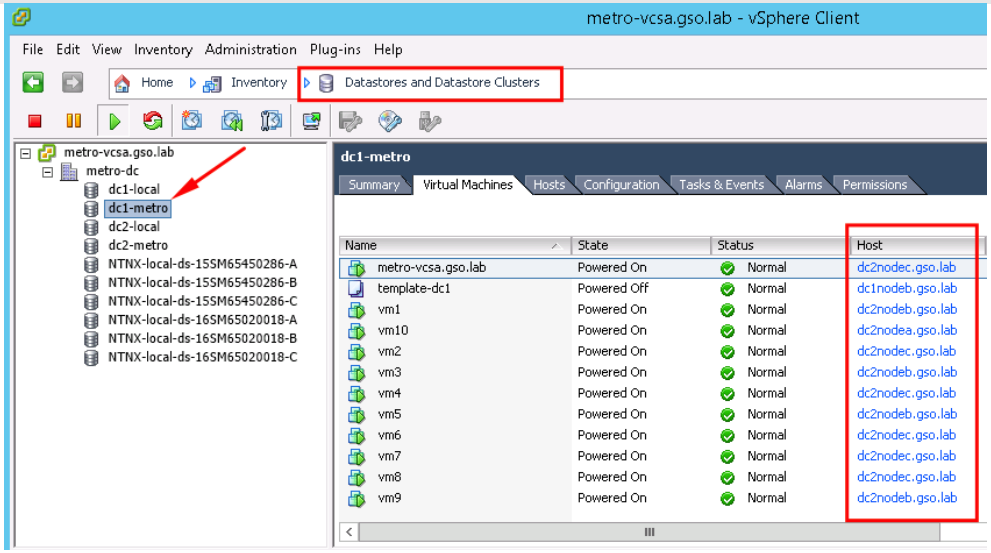
4.



Modifier le « **Cluster Host Group** » et sélectionnez le groupe d'hôtes de *dc2*, puis cliquez sur **OK**.

5. 

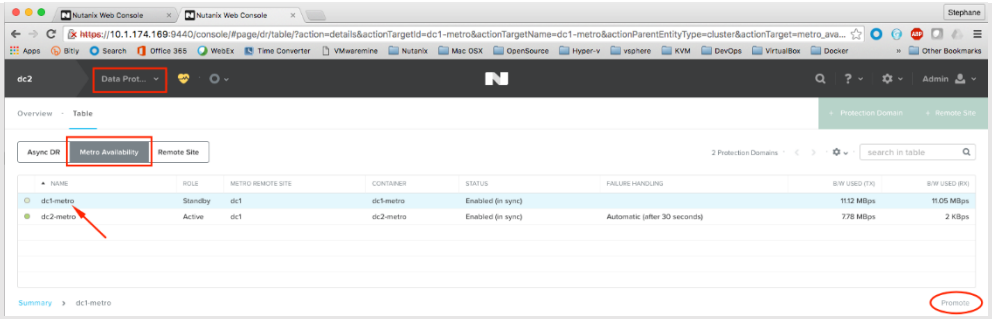
Nous pouvons à présent forcer DRS à migrer les VMs vers les hôtes du site cible *dc2*.

6. 

Il faut à présent vérifier que toutes les machines virtuelles ont bien été migrées en sélectionnant la banque de données *dc1-metro* dans le client vSphere.

Si certaines VMs n'ont pas été migrées par DRS, il vous faudra les migrer manuellement à l'aide d'un vMotion.

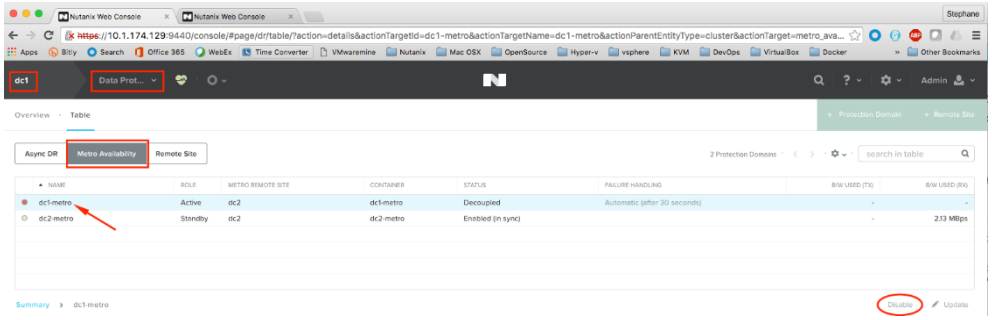
Si la banque de données contient des modèles de machines virtuelles, il n'est pas nécessaire de les migrer.

7. 

Après avoir confirmé que les VMs tournent bien désormais sur *dc2*, nous pouvons à présent effectuer la promotion du « protection domain » *dc1-metro* sur le site cible *dc2* dans l'interface Prism.
Notez que Prism vous demandera de confirmer l'opération en vous faisant écrire **PROMOTE** en toute lettre.

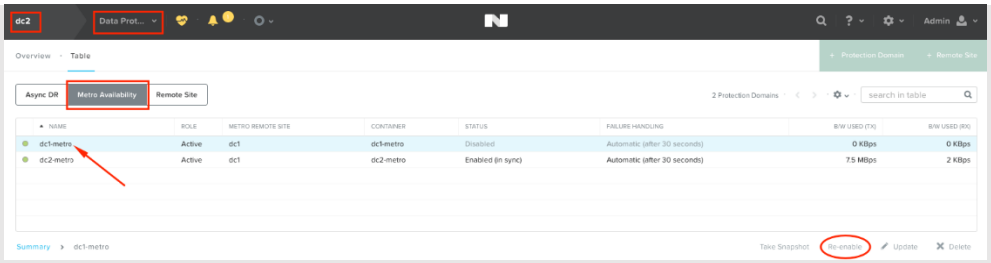
Note

Si vous souhaitez réduire la quantité de données à répliquer lorsque la réplication sera réactivée, vous pouvez prendre un snapshot manuel depuis le site source avant de faire la promotion du « protection domain »

8. 

Une fois la promotion du « protection domain » effectuée sur le site cible *dc2*, nous devons désactiver la réplication depuis le site source *dc1.*

9.



Stop

Attention, cette action écrasera les données du container *dc1-metro* sur le site source *dc1*. Assurez-vous de ne pas avoir de VMs sur le container *dc1-metro* qui tournent sur des hôtes du site source *dc1*. Si c'est le cas, utilisez le Storage vMotion pour les évacuer vers un autre container avant de continuer.

Nous pouvons à présent activer la réplication synchrone depuis le site cible *dc2* vers le site source *dc1*.

Note

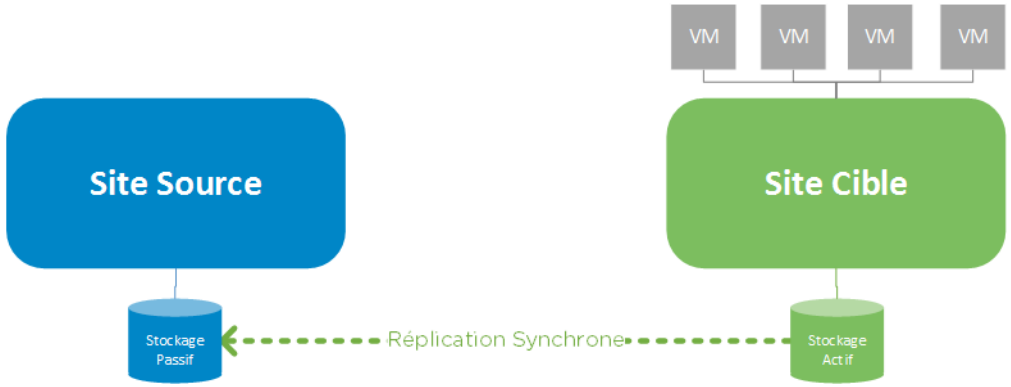
Même si le site *dc1* doit être arrêté, il est préférable d'avoir la réplication active afin qu'elle puisse redémarrer dès que le site sera à nouveau disponible.

Nous en avons fini avec la procédure de bascule planifiée !

Retour arrière (planifié)

Utilisez cette procédure pour faire un retour arrière après une bascule planifiée.

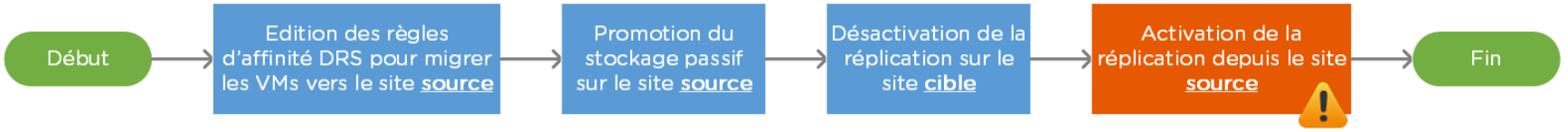
La situation initiale est la suivante :



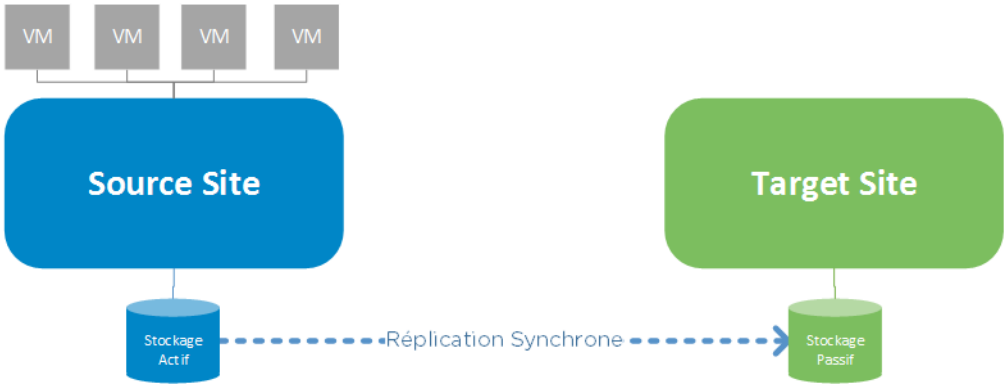
Notons que :

1. Les VMs tournent sur le site cible (dc2) où le stockage est actif.
2. La réplication synchrone s'effectue depuis le site cible (*dc2*) vers le site source (*dc1*)

Le flux de travail est le suivant :



Une fois la procédure déroulée l'arrêté de situation sera le suivant :

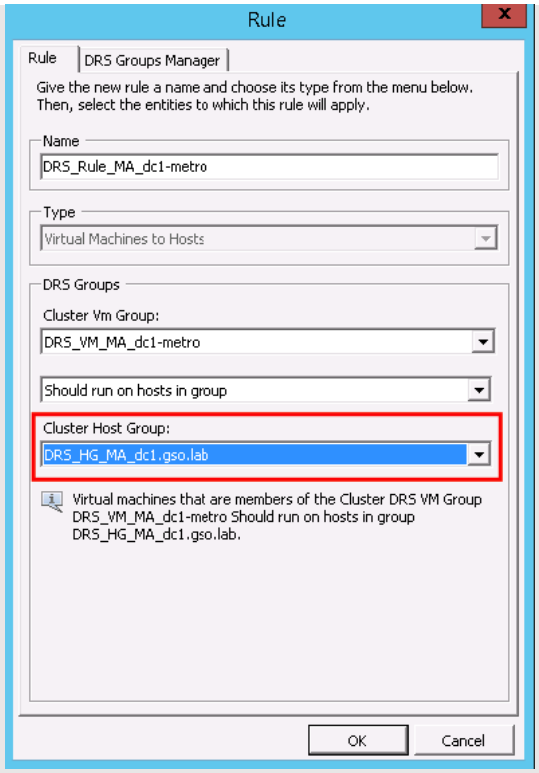


Notons que :

1. Les VMs tourneront sur le site source (dc1)
2. Le stockage sera actif sur le site source (dc1)
3. La réplication synchrone s'effectuera depuis le site source (dc1) vers le site cible (dc2)

Pour le détail de la procédure ci-dessous, notons que :

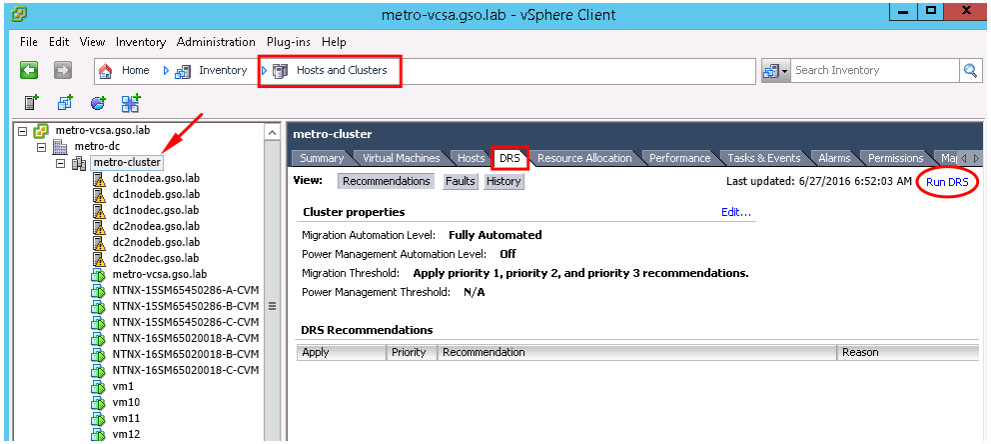
1. Le site source est *dc1*
2. Le site cible est *dc2*
3. Le « container » (entité logique de stockage) est *dc1-metro*
4. Le « protection domain » (entité logique de réplication) est également appelé *dc1-metro*
5. Les VMs *vm1* à *vm10* tournent sur *dc1*
6. Le cluster vSphere est *metro-cluster*
7. Certaines captures d'écran montrent d'autres « containers » et « protection domains » car dc2 est également répliqué vers dc1, comme c'est le cas dans la plupart des environnements de production.

- 

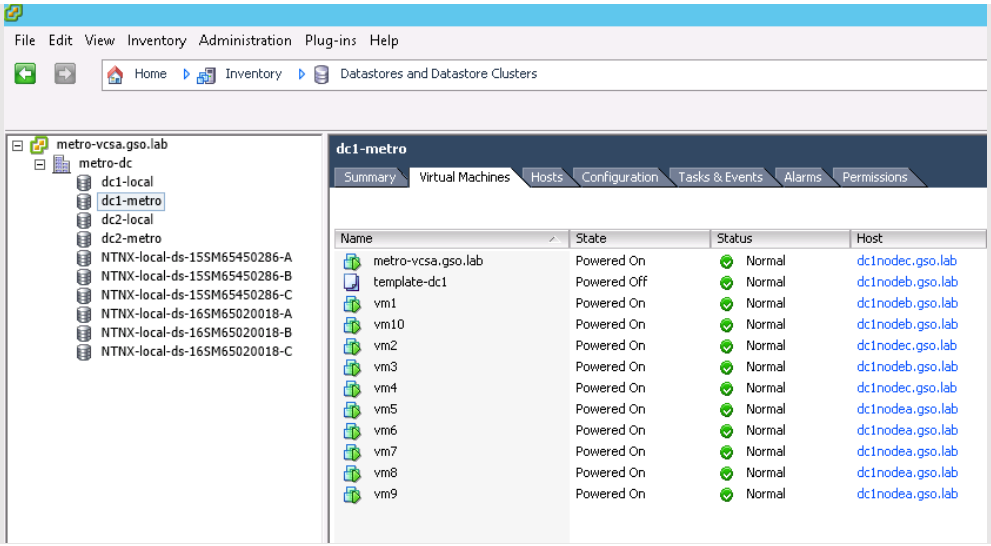
Stop

Avant de continuer, assurez-vous que dans Prism que le « protection domain » *dc1-metro* est dans l'état « **Enabled (in sync)** »

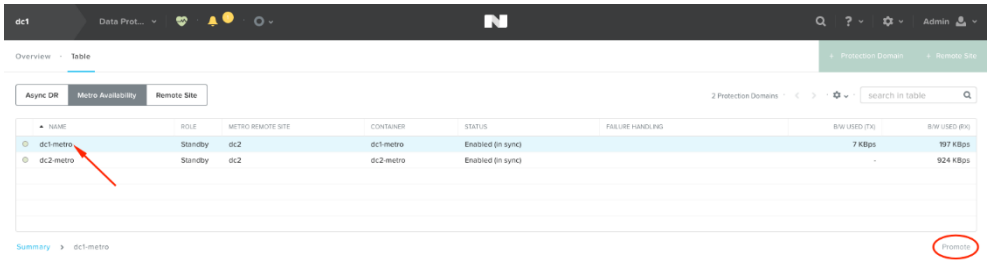
Commencez par éditer la règle DRS de *dc1-metro* et sélectionnez le groupe d'hôte du site source *dc1*.

- 

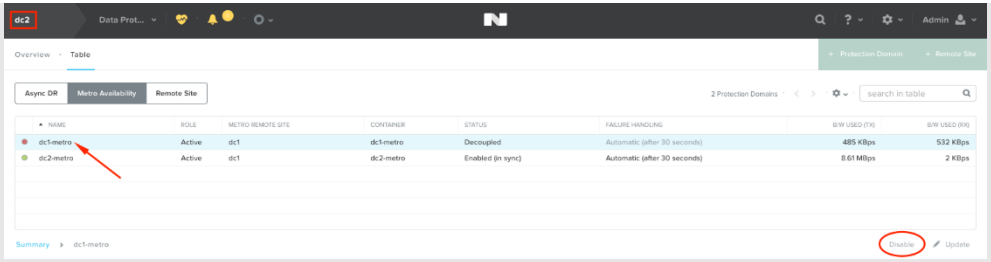
Vous pouvez à présent forcer DRS à appliquer les recommandations et migrer les VMs vers les hôtes du site source *dc1*.

- 

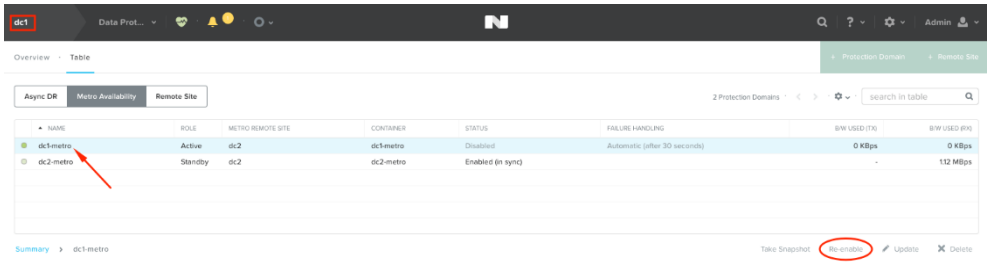
Sélectionnez la banque de données du site source *dc1* et vérifiez que toutes les machines virtuelles ont bien été migrées vers *dc1*. Si ce n'est pas le cas, migrez manuellement les VMs restantes.

- 

  Effectuez la promotion du « protection domain » *dc1*-metro sur le site source *dc1*.

- 

  Désactivez la réplication du « protection domain » *dc1*-metro sur le site cible *dc2*.

- 

Stop

Attention, cette action écrasera les données du container *dc1*-metro sur le site cible *dc2*. Assurez-vous de ne pas avoir de VMs sur le container *dc1-metro* qui tournent sur des hôtes du site cible *dc2*. Si c'est le cas, utilisez le Storage vMotion pour les évacuer vers un autre container avant de continuer.

Vous pouvez désormais réactiver la réplication du « protection domain » *dc1*-metro depuis le site source vers le site cible.

Vous avez fini de dérouler la procédure de retour arrière !

Procédure #2 : Bascule non-planifiée (Sinistre)

Introduction

Vous devez effectuer la procédure de bascule non-planifiée lorsqu'un sinistre intervient sur l'un de vos sites et que celui-ci n'est plus du tout disponible, c'est-à-dire que les hôtes de ce site sont arrêtés. L'objectif de cette procédure est de s'assurer que les machines virtuelles et leur stockage sont récupérés sur le site restant.

Lorsque le site ayant subi un sinistre sera à nouveau disponible, vous devrez effectuer la procédure de retour arrière.

Cette procédure utilise la terminologie suivante :

- Le site ayant subi un sinistre est appelé le site primaire.
- Le site restant sur lequel vous voulez récupérer vos machines virtuelles et leur stockage est appelé le site de récupération.

Pour la procédure de retour arrière, la bascule sera depuis le site de récupération vers le site primaire.

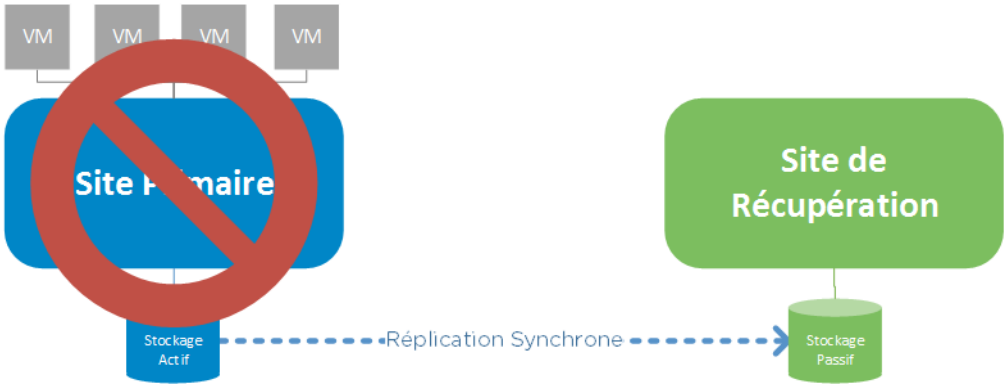
Chaque procédure est décrite en détail mais contient aussi un schéma de :

1. Situation initiale : ce schéma montre la situation de départ. Où sont les VMs, où le stockage est actif et dans quel sens s'effectue la réplication.
2. Un flux de travail : il décrit les grandes étapes de la procédure. Les actions destructrices sont indiquées en orange et labélisées avec un point d'exclamation dans un triangle. Avant d'effectuer ces actions, assurez-vous que vos machines virtuelles sont sur le bon site, sinon vous risquez de perdre des données et une interruption de service.

3. Un arrêté de situation : ce schéma montre où sont les VMs, le stockage actif et le sens de la réplication une fois que la procédure a été déroulée.

Bascule du site primaire vers le site secondaire

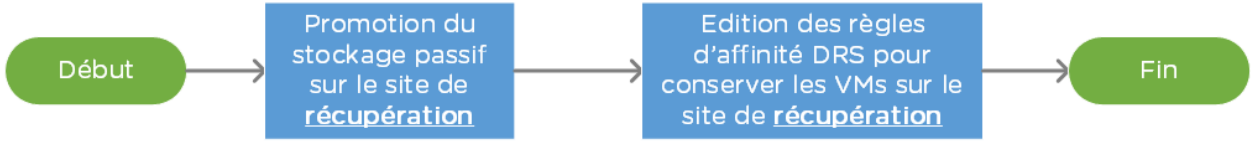
La situation initiale est la suivante :



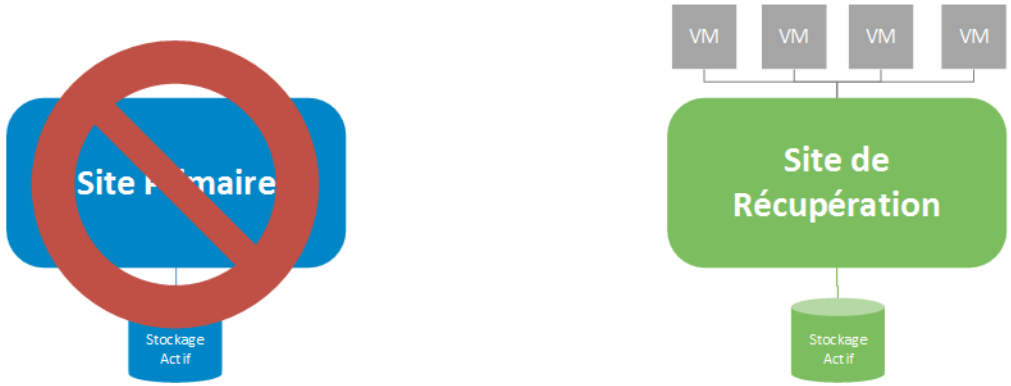
Notons que :

1. Le site primaire où tournaient les VMs et qui détenait le stockage actif a subi un sinistre et n'est plus disponible
2. La réplication synchrone s'effectuait du site primaire vers le site secondaire

Le flux de travail est le suivant :



Une fois la procédure déroulée, l'arrêté de situation sera le suivant :



Notons que :

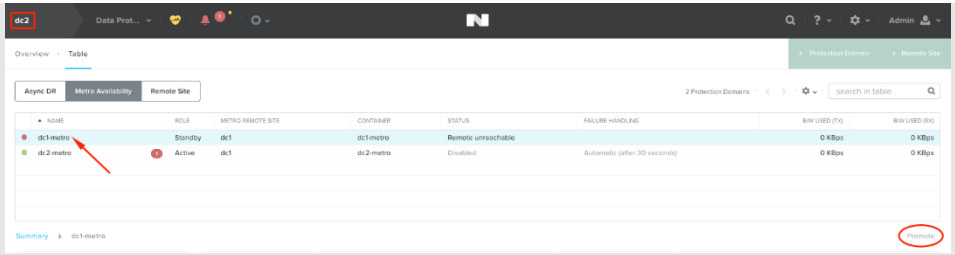
1. Les VMs tourneront sur le site de récupération
2. Le stockage sera actif sur le site de récupération
3. La réplication ne sera pas active étant donné que le site primaire n'est pas disponible

Pour le détail de la procédure ci-dessous, notons que :

1. Le site primaire est *dc1*
2. Le site de récupération est *dc2*
3. Le « container » (entité logique de stockage) est *dc1-metro*
4. Le « protection domain » (entité logique de réplication) est également appelé *dc1-metro*
5. Les VMs *vm1* à *vm10* tournent sur *dc1*
6. Le cluster vSphere est *metro-cluster*
7. Certaines captures d'écran montrent d'autres « containers » et « protection domains » car dc2 est également répliqué vers dc1, comme c'est le cas dans la plupart des environnements de production.

Stop

Avant de démarrer la procédure de bascule non planifiée, assurez-vous que le site primaire a bien subi un désastre et restera indisponible pendant un temps supérieur à votre RTO (recovery time objective). Dans le cas contraire et si certaines VMs continuent à tourner sur le site primaire et demeurent accessibles sur le réseau, vous pourrez éventuellement subir des pertes de données.

1. 

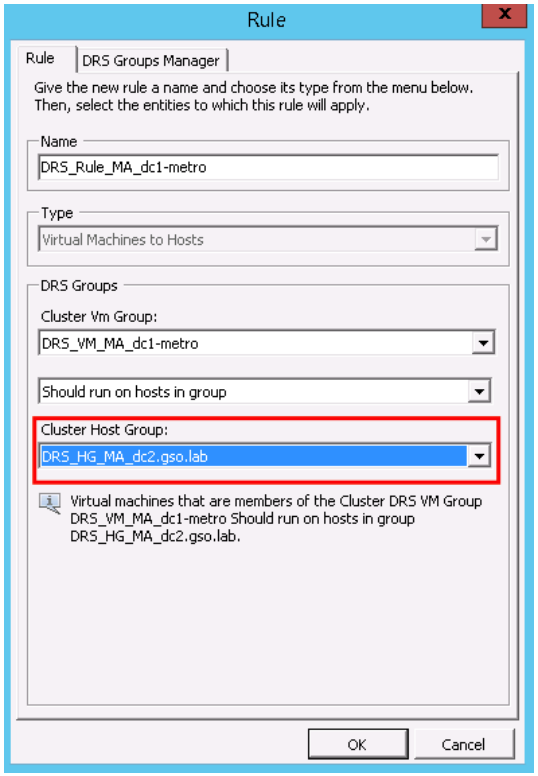
Commencez par effectuer la promotion du « protection domain » *dc1-metro* dans Prism sur le site de récupération *dc2*.

Attendez que VMware vSphere HA (high availability) redémarre les VMs du site primaire sur le site secondaire. Notez que cela peut prendre plusieurs minutes.

Warning

Soyez patient et ne tentez pas de redémarrer vous-même les VMs au risque de perturber HA.

2.



Editez à présent la règle DRS du container *dc1-metro* afin de garder les VMs de cette banque de données sur les hôtes du site de récupération *dc2*. Notez que vous aurez besoin de la VM vCenter pour éditer les règles DRS. Si cette VM était sur le site primaire, il vous faudra attendre qu'elle soit redémarrée par HA.

L'édition de la règle DRS est importante car si le site primaire *dc1* redevenait disponible prématurément, DRS essaierait de déplacer des VMs vers les hôtes de *dc1* qui détiendra lui aussi une copie active (mais désynchronisée) du stockage.

Note

Si vous n'avez pas eu le temps d'éditer les règles DRS avant que le site primaire redevienne disponible, vous devrez immédiatement arrêter les VMs, puis effectuer la procédure de retour arrière.

Vous avez terminé !

Retour arrière (sinistre)

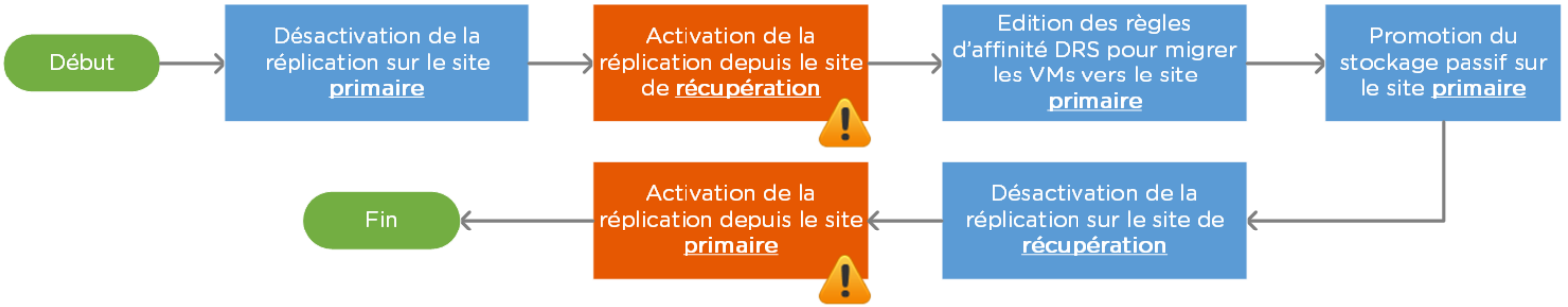
N'utilisez cette procédure que pour effectuer un retour arrière suite à une bascule non-planifiée.

La situation initiale est la suivante :

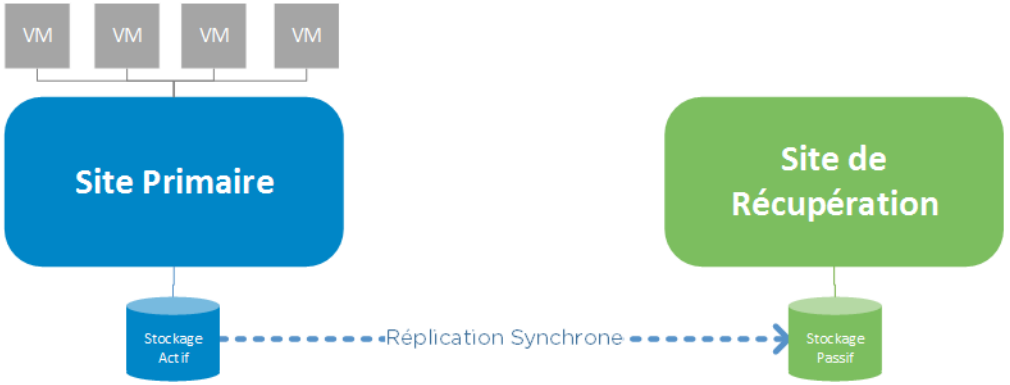
Notons que :

1. Les VMs tournent sur le site de récupération *dc2* où le stockage est actif.
2. Le site primaire *dc1* est à nouveau disponible et détient également une copie active (mais obsolète) du stockage.
3. La réplication synchrone n'est pas activée et le « protection domain » dans un état « decoupled »

Le flux de travail est le suivant :



Une fois la procédure déroulée, l'arrêté de situation sera le suivant :



Notons que :

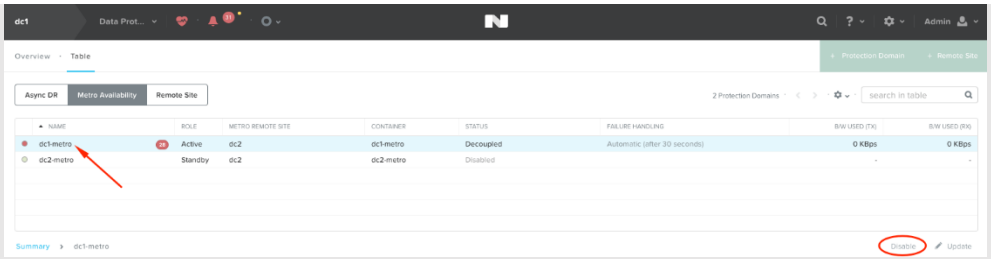
1. Les VMs tourneront sur le site primaire *dc1*
2. Le stockage sera actif sur le site primaire *dc1* et passif (standby) sur le site de récupération *dc2*
3. La réplication synchrone s'effectuera depuis le site primaire *dc1* vers le site de récupération *dc2*

Pour le détail de la procédure ci-dessous, notons que :

1. Le site primaire est *dc1*
2. Le site de récupération est *dc2*

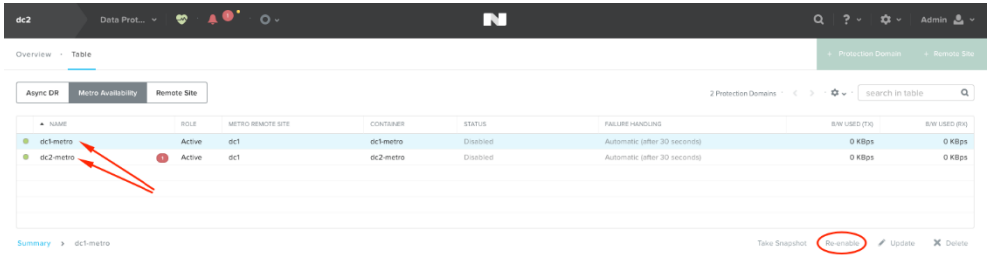
3. Le « container » (entité logique de stockage) est *dc1-metro*
4. Le « protection domain » (entité logique de réplication) est également appelé *dc1-metro*
5. Les VMs *vm1* à *vm10* tournent sur *dc1*
6. Le cluster vSphere est *metro-cluster*
7. Certaines captures d'écran montrent d'autres « containers » et « protection domains » car dc2 est également répliqué vers dc1, comme c'est le cas dans la plupart des environnements de production.

1.



Commencez par désactiver la réplication du « protection domain » *dc1-metro* dans Prism sur le site primaire *dc1* dès que ce dernier est à nouveau opérationnel.

2.



Stop

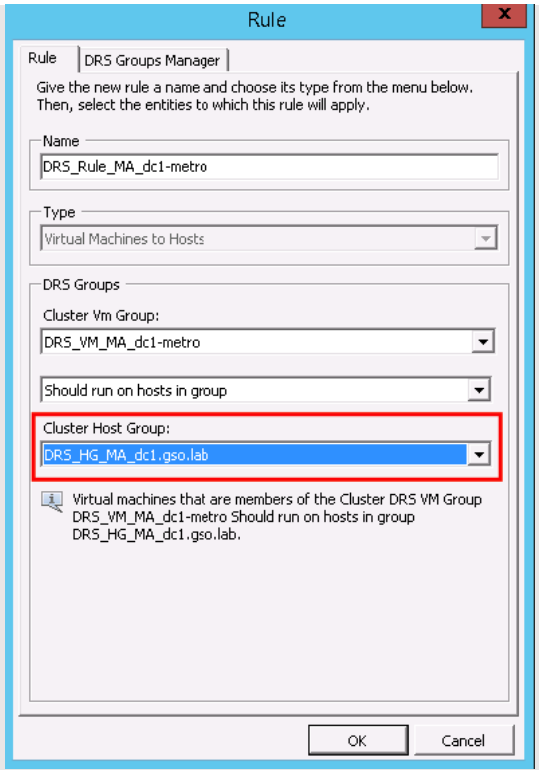
Attention, cette action écrasera les données du container *dc1-metro* sur le site primaire *dc1*. Assurez-vous donc de ne pas avoir de VMs sur le container *dc1-metro* qui tournent sur des hôtes du site primaire *dc1*.

Activez à présent la réplication du « protection domain » *dc1*-metro depuis le site de récupération *dc2* vers le site primaire *dc1*.

Si vous avez un autre « protection domain » qui était répliqué de *dc2* vers *dc1*, vous devrez également le réactiver.

Une fois la réplication terminée, cela signifie que toutes les données qui ont été modifiées depuis le début du sinistre sont maintenant à jour également sur le site primaire.
Nous sommes donc prêts à migrer les VMs vers *dc1*.

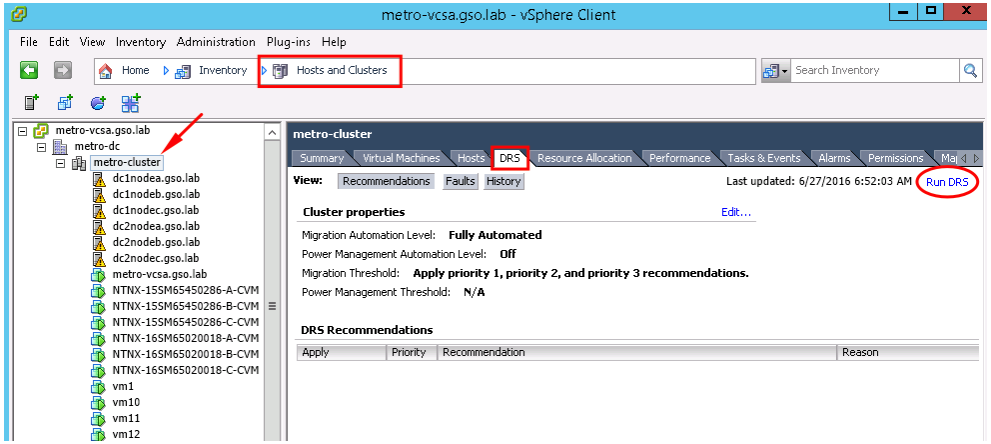
3.



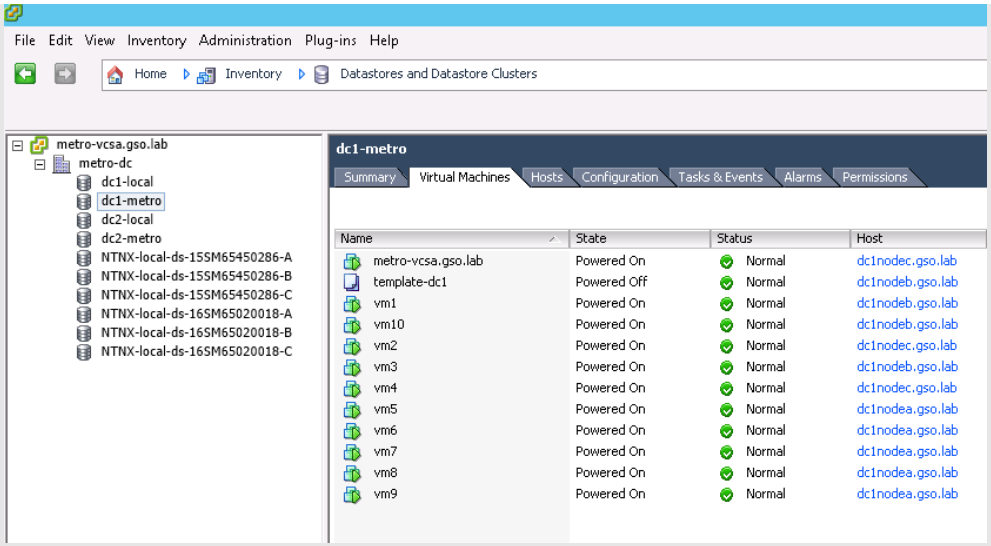
Stop

Avant de continuer, assurez-vous que dans Prism que le « protection domain » *dc1-metro* est dans l'état « **Enabled (in sync)** »

Editez la règle DRS *dc1-metro* et sélectionnez le groupe d'hôte du site primaire *dc1*.

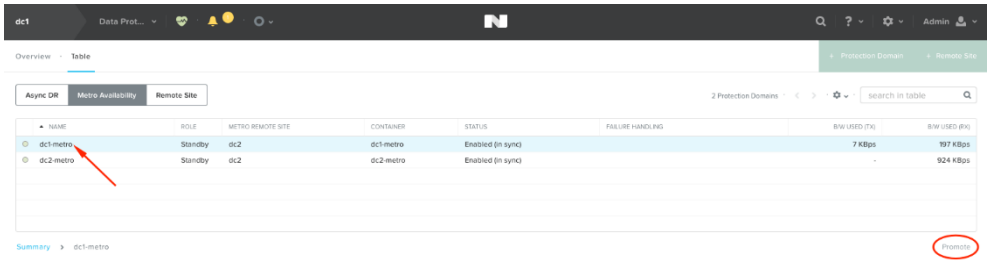
4. 

Forcez DRS à appliquer les recommandations afin que les VMs du container *dc1-metro* soient migrées vers les hôtes du site primaire *dc1*.

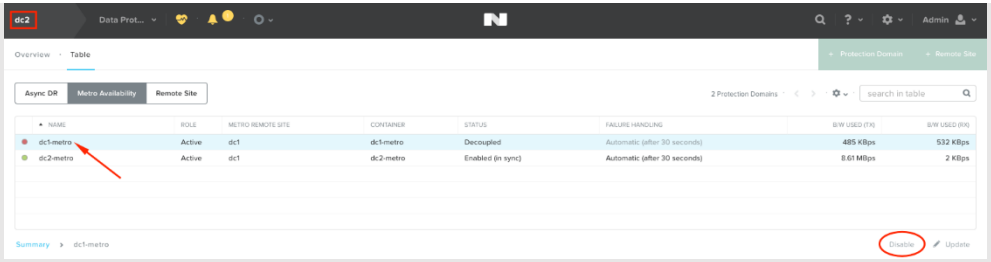
5. 

Sélectionnez la banque de données *dc1-metro* et assurez-vous que l'ensemble des machines virtuelles tournent bien sur des hôtes du site primaire *dc1*.

Si certaines VMs n'ont pas été migrées par DRS, vous devrez les migrer manuellement à l'aide de vMotion vers des hôtes du site primaire *dc1*.

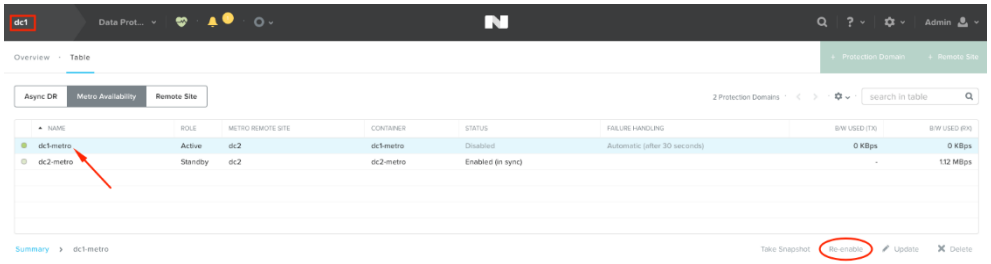
6. 

Effectuez la promotion du « protection domain » *dc1*-metro sur le site primaire *dc1*.

7. 

Désactivez la réplication du « protection domain » *dc1*-metro sur le site de récupération *dc2*.

8.



Stop

Attention, cette action écrasera les données du container *dc1-letro* sur le site de récupération *dc2*. Assurez-vous de ne pas avoir de VMs sur le container *dc1-metro* qui tournent sur des hôtes du site de récupération *dc2*. Si c'est le cas, utilisez le Storage vMotion pour les évacuer vers un autre container avant de continuer.

Pour finir, activez la réplication du « protection domain » *dc1-metro* dans Prism depuis le site primaire *dc1* vers le site de récupération *dc2*.

Vous avez terminé !

Script PowerShell pour automatiser la création des règles d'affinité DRS

Le script ci-dessous peut être utilisé pour automatiser la création ou la mise à jour des règles d'affinité DRS d'un cluster vSphere sur Nutanix avec la fonctionnalité Metro Availability activée.

```
<#
.SYNOPSIS
  This script is used to create DRS affinity groups and rules
based on the Nutanix Metro Availability setup of a vSphere
cluster.
.DESCRIPTION
  The script will look at the Metro Availability setup for a
pair of given Nutanix clusters and will create DRS affinity
groups and rules so that VMs will run on hosts which hold the
active copy of a given replicated datastore. This is to avoid
I/O going over two sites in normal conditions.  If DRS groups
and rules already exist that match the naming convention used
in this script, then it will update those groups and rules
(unless you use the -noruleupdate switch in which case only
groups will be updated).  This script requires having both the
Nutanix cmdlets and PowerCLI installed.
.PARAMETER help
  Displays a help message (seriously, what did you think this
was )
.PARAMETER history
  Displays a release history for this script (provided the
editors were smart enough to document this...)
.PARAMETER log
  Specifies that you want the output messages to be written in
a log file as well as on the screen.
.PARAMETER debugme
  Turns off SilentlyContinue on unexpected error messages.
.PARAMETER ntnx_cluster1
  First Nutanix cluster fully qualified domain name or IP
address.
.PARAMETER ntnx_cluster2
  Second Nutanix cluster fully qualified domain name or IP
address.
.PARAMETER username
  Username used to connect to the Nutanix clusters.
.PARAMETER password
  Password used to connect to the Nutanix clusters.
.PARAMETER vcenter
  Hostname or IP address of the vCenter Server.
.PARAMETER noruleupdate
  Use this switch if you do NOT want to update DRS rules. Only
groups will be updated. This can be useful when using the
script within the context of a failback.
.EXAMPLE
  Create DRS affinity groups and rules for ntnxc1 and ntnxc2 on
vcenter1:
  PS> .\add-DRSAffinityRulesForMA.ps1 -ntnx_cluster1
ntnxc1.local -ntnx_cluster2 ntnxc2.local -username admin -
password nutanix/4u -vcenter vcenter1.local
.LINK
  http://www.nutanix.com/services
.NOTES
  Author: Stephane Bourdeaud (sbourdeaud@nutanix.com)
  Revision: June 22nd 2016
#>

#####################################
##   parameters and initial setup   ##
#####################################
#let's start with some command line parsing
Param
(
    #[parameter(valuefrompipeline = $true, mandatory = $true)]
[PSObject]$myParam1,
    [parameter(mandatory = $false)] [switch]$help,
    [parameter(mandatory = $false)] [switch]$history,
    [parameter(mandatory = $false)] [switch]$log,
    [parameter(mandatory = $false)] [switch]$debugme,
    [parameter(mandatory = $false)] [string]$ntnx_cluster1,
    [parameter(mandatory = $false)] [string]$ntnx_cluster2,
    [parameter(mandatory = $false)] [string]$username,
    [parameter(mandatory = $false)] [string]$password,
    [parameter(mandatory = $false)] [string]$vcenter,
    [parameter(mandatory = $false)] [switch]$noruleupdate

)

# get rid of annoying error messages
```

```
if (!$debugme) {$ErrorActionPreference = "SilentlyContinue"}

########################
##   main functions   ##
########################

#this function is used to output log data
Function OutputLogData
{
          #input: log category, log message
          #output: text to standard output
<#
.SYNOPSIS
  Outputs messages to the screen and/or log file.
.DESCRIPTION
  This function is used to produce screen and log output which
is categorized, time stamped and color coded.
.NOTES
  Author: Stephane Bourdeaud
.PARAMETER myCategory
  This the category of message being outputed. If you want
color coding, use either "INFO", "WARNING", "ERROR" or "SUM".
.PARAMETER myMessage
  This is the actual message you want to display.
.EXAMPLE
  PS> OutputLogData -mycategory "ERROR" -mymessage "You must
specify a cluster name!"
#>
          param
          (
                    [string] $category,
                    [string] $message
          )

    begin
    {
             $myvarDate = get-date
             $myvarFgColor = "Gray"
             switch ($category)
             {
                       "INFO" {$myvarFgColor = "Green"}
                       "WARNING" {$myvarFgColor = "Yellow"}
                       "ERROR" {$myvarFgColor = "Red"}
                       "SUM" {$myvarFgColor = "Magenta"}
             }
    }

    process
    {
                       Write-Host -ForegroundColor $myvarFgColor
"$myvarDate [$category] $message"
                       if ($log) {Write-Output "$myvarDate [$category]
$message" >>$myvarOutputLogFile}
    }

    end
    {
        Remove-variable category
        Remove-variable message
        Remove-variable myvarDate
        Remove-variable myvarFgColor
    }
}#end function OutputLogData

#this function is used to create a DRS host group
Function New-DrsHostGroup
{
<#
.SYNOPSIS
  Creates a new DRS host group
.DESCRIPTION
  This function creates a new DRS host group in the DRS Group
Manager
.NOTES
  Author: Arnim van Lieshout
.PARAMETER VMHost
  The hosts to add to the group. Supports objects from the
pipeline.
.PARAMETER Cluster
  The cluster to create the new group on.
.PARAMETER Name
  The name for the new group.
.EXAMPLE
  PS> Get-VMHost ESX001,ESX002 | New-DrsHostGroup -Name
"HostGroup01" -Cluster CL01
.EXAMPLE
  PS> New-DrsHostGroup -Host ESX001,ESX002 -Name "HostGroup01"
-Cluster (Get-CLuster CL01)
#>

    Param(
        [parameter(valuefrompipeline = $true, mandatory =
$true,
        HelpMessage = "Enter a host entity")]
            [PSObject]$VMHost,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a cluster entity")]
            [PSObject]$Cluster,
```

```
        [parameter(mandatory = $true,
        HelpMessage = "Enter a name for the group")]
            [String]$Name)

    begin {
        switch ($Cluster.gettype().name) {
            "String" {$cluster = Get-Cluster $cluster | Get-
View}
            "ClusterImpl" {$cluster = $cluster | Get-View}
            "Cluster" {}
            default {throw "No valid type for parameter -
Cluster specified"}
        }
        $spec = New-Object VMware.Vim.ClusterConfigSpecEx
        $group = New-Object VMware.Vim.ClusterGroupSpec
        $group.operation = "add"
        $group.Info = New-Object VMware.Vim.ClusterHostGroup
        $group.Info.Name = $Name
    }

    Process {
        switch ($VMHost.gettype().name) {
            "String[]" {Get-VMHost -Name $VMHost |
%{$group.Info.Host += $_.Extensiondata.MoRef}}
            "String" {Get-VMHost -Name $VMHost |
%{$group.Info.Host += $_.Extensiondata.MoRef}}
            "VMHostImpl" {$group.Info.Host +=
$VMHost.Extensiondata.MoRef}
            "HostSystem" {$group.Info.Host += $VMHost.MoRef}
            default {throw "No valid type for parameter -VMHost
specified"}
        }
    }

    End {
        if ($group.Info.Host) {
            $spec.GroupSpec += $group

$cluster.ReconfigureComputeResource_Task($spec,$true) | Out-
Null
        }
        else {
            throw "No valid hosts specified"
        }
    }
}

#this function is used to create a DRS VM group
Function New-DrsVmGroup
{
<#
.SYNOPSIS
  Creates a new DRS VM group
.DESCRIPTION
  This function creates a new DRS VM group in the DRS Group
Manager
.NOTES
  Author: Arnim van Lieshout
.PARAMETER VM
  The VMs to add to the group. Supports objects from the
pipeline.
.PARAMETER Cluster
  The cluster to create the new group on.
.PARAMETER Name
  The name for the new group.
.EXAMPLE
  PS> Get-VM VM001,VM002 | New-DrsVmGroup -Name "VmGroup01" -
Cluster CL01
.EXAMPLE
  PS> New-DrsVmGroup -VM VM001,VM002 -Name "VmGroup01" -Cluster
(Get-CLuster CL01)
#>

    Param(
        [parameter(valuefrompipeline = $true, mandatory =
$true,
        HelpMessage = "Enter a vm entity")]
            [PSObject]$VM,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a cluster entity")]
            [PSObject]$Cluster,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a name for the group")]
            [String]$Name)

    begin {
        switch ($Cluster.gettype().name) {
            "String" {$cluster = Get-Cluster $cluster | Get-
View}
            "ClusterImpl" {$cluster = $cluster | Get-View}
            "Cluster" {}
            default {throw "No valid type for parameter -
Cluster specified"}
        }
        $spec = New-Object VMware.Vim.ClusterConfigSpecEx
        $group = New-Object VMware.Vim.ClusterGroupSpec
        $group.operation = "add"
        $group.Info = New-Object VMware.Vim.ClusterVmGroup
```

```
        $group.Info.Name = $Name
    }

    Process {
        switch ($VM.gettype().name) {
            "String[]" {Get-VM -Name $VM | %{$group.Info.VM +=
$_.Extensiondata.MoRef}}
            "String" {Get-VM -Name $VM | %{$group.Info.VM +=
$_.Extensiondata.MoRef}}
            "VirtualMachineImpl" {$group.Info.VM +=
$VM.Extensiondata.MoRef}
            "VirtualMachine" {$group.Info.VM += $VM.MoRef}
            default {throw "No valid type for parameter -VM
specified"}
        }
    }

    End {
        if ($group.Info.VM) {
            $spec.GroupSpec += $group

$cluster.ReconfigureComputeResource_Task($spec,$true) | Out-
Null
        }
        else {
            throw "No valid VMs specified"
        }
    }
}

#this function is used to create a VM to host DRS rule
Function New-DRSVMToHostRule
{
<#
.SYNOPSIS
  Creates a new DRS VM to host rule
.DESCRIPTION
  This function creates a new DRS vm to host rule
.NOTES
  Author: Arnim van Lieshout
.PARAMETER VMGroup
  The VMGroup name to include in the rule.
.PARAMETER HostGroup
  The VMHostGroup name to include in the rule.
.PARAMETER Cluster
  The cluster to create the new rule on.
.PARAMETER Name
  The name for the new rule.
.PARAMETER AntiAffine
  Switch to make the rule an AntiAffine rule. Default rule type
is Affine.
.PARAMETER Mandatory
  Switch to make the rule mandatory (Must run rule). Default
rule is not mandatory (Should run rule)
.EXAMPLE
  PS> New-DrsVMToHostRule -VMGroup "VMGroup01" -HostGroup
"HostGroup01" -Name "VMToHostRule01" -Cluster CL01 -AntiAffine
-Mandatory
#>

    Param(
        [parameter(mandatory = $true,
        HelpMessage = "Enter a VM DRS group name")]
            [String]$VMGroup,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a host DRS group name")]
            [String]$HostGroup,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a cluster entity")]
            [PSObject]$Cluster,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a name for the group")]
            [String]$Name,
            [Switch]$AntiAffine,
            [Switch]$Mandatory)

    switch ($Cluster.gettype().name) {
        "String" {$cluster = Get-Cluster $cluster | Get-View}
        "ClusterImpl" {$cluster = $cluster | Get-View}
        "Cluster" {}
        default {throw "No valid type for parameter -Cluster
specified"}
    }

    $spec = New-Object VMware.Vim.ClusterConfigSpecEx
    $rule = New-Object VMware.Vim.ClusterRuleSpec
    $rule.operation = "add"
    $rule.info = New-Object VMware.Vim.ClusterVmHostRuleInfo
    $rule.info.enabled = $true
    $rule.info.name = $Name
    $rule.info.mandatory = $Mandatory
    $rule.info.vmGroupName = $VMGroup
    if ($AntiAffine) {
        $rule.info.antiAffineHostGroupName = $HostGroup
    }
    else {
        $rule.info.affineHostGroupName = $HostGroup
    }
```

```
    $spec.RulesSpec += $rule
    $cluster.ReconfigureComputeResource_Task($spec,$true) |
Out-Null
}

#this function is used to edit an existing DRS rule
Function Update-DrsVMGroup
{
<#
.SYNOPSIS
Update DRS VM group with a new collection of VM´s

.DESCRIPTION
Use this function to update the ClusterVMgroup with VMs that
are sent in by parameters

.PARAMETER  xyz

.NOTES
Author: Niklas Akerlund / RTS (most of the code came from
http://communities.vmware.com/message/1667279 @LucD22 and
GotMoo)
Date: 2012-06-28
#>
          param
    (
              $cluster,
              $VMs,
              $groupVMName
    )

    $cluster = Get-Cluster $cluster
    $spec = New-Object VMware.Vim.ClusterConfigSpecEx
    $groupVM = New-Object VMware.Vim.ClusterGroupSpec
    #Operation edit will replace the contents of the
GroupVMName with the new contents seleced below.
    $groupVM.operation = "edit"

    $groupVM.Info = New-Object VMware.Vim.ClusterVmGroup
    $groupVM.Info.Name = $groupVMName

    Get-VM $VMs | %{$groupVM.Info.VM += $_.Extensiondata.MoRef}
    $spec.GroupSpec += $groupVM

    #Apply the settings to the cluster

$cluster.ExtensionData.ReconfigureComputeResource($spec,$true)
}
```

```
#this function is used to edit an existing DRS rule
Function Update-DrsHostGroup
{
<#
.SYNOPSIS
Update DRS Host group with a new collection of Hosts

.DESCRIPTION
Use this function to update the ClusterHostgroup with Hosts
that are sent in by parameters

.PARAMETER  xyz

.NOTES
Author: Niklas Akerlund / RTS (most of the code came from
http://communities.vmware.com/message/1667279 @LucD22 and
GotMoo)
Date: 2012-06-28
#>
          param
    (
              $cluster,
              $Hosts,
              $groupHostName
    )

    $cluster = Get-Cluster $cluster
    $spec = New-Object VMware.Vim.ClusterConfigSpecEx
    $groupHost = New-Object VMware.Vim.ClusterGroupSpec
    #Operation edit will replace the contents of the
GroupVMName with the new contents seleced below.
    $groupHost.operation = "edit"

    $groupHost.Info = New-Object VMware.Vim.ClusterHostGroup
    $groupHost.Info.Name = $groupHostName

    Get-VMHost $Hosts | %{$groupHost.Info.Host +=
$_.Extensiondata.MoRef}
    $spec.GroupSpec += $groupHost

    #Apply the settings to the cluster

$cluster.ExtensionData.ReconfigureComputeResource($spec,$true)
}

#this function is used to create a VM to host DRS rule
Function Update-DRSVMToHostRule
{
<#
```

```
.SYNOPSIS
  Creates a new DRS VM to host rule
.DESCRIPTION
  This function creates a new DRS vm to host rule
.NOTES
  Author: Arnim van Lieshout
.PARAMETER VMGroup
  The VMGroup name to include in the rule.
.PARAMETER HostGroup
  The VMHostGroup name to include in the rule.
.PARAMETER Cluster
  The cluster to create the new rule on.
.PARAMETER Name
  The name for the new rule.
.PARAMETER AntiAffine
  Switch to make the rule an AntiAffine rule. Default rule type
is Affine.
.PARAMETER Mandatory
  Switch to make the rule mandatory (Must run rule). Default
rule is not mandatory (Should run rule)
.EXAMPLE
  PS> New-DrsVMToHostRule -VMGroup "VMGroup01" -HostGroup
"HostGroup01" -Name "VMToHostRule01" -Cluster CL01 -AntiAffine
-Mandatory
#>

    Param(
        [parameter(mandatory = $true,
        HelpMessage = "Enter a VM DRS group name")]
            [String]$VMGroup,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a DRS rule key")]
            [String]$RuleKey,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a DRS rule uuid")]
            [String]$RuleUuid,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a host DRS group name")]
            [String]$HostGroup,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a cluster entity")]
            [PSObject]$Cluster,
        [parameter(mandatory = $true,
        HelpMessage = "Enter a name for the group")]
            [String]$Name,
            [Switch]$AntiAffine,
            [Switch]$Mandatory)

    switch ($Cluster.gettype().name) {
        "String" {$cluster = Get-Cluster $cluster | Get-View}
        "ClusterImpl" {$cluster = $cluster | Get-View}
        "Cluster" {}
        default {throw "No valid type for parameter -Cluster
specified"}
    }

    $spec = New-Object VMware.Vim.ClusterConfigSpecEx
    $rule = New-Object VMware.Vim.ClusterRuleSpec
    $rule.operation = "edit"
    $rule.info = New-Object VMware.Vim.ClusterVmHostRuleInfo
    $rule.info.enabled = $true
    $rule.info.name = $Name
    $rule.info.mandatory = $Mandatory
    $rule.info.vmGroupName = $VMGroup
    $rule.info.Key = $RuleKey
    $rule.info.RuleUuid = $RuleUuid
    if ($AntiAffine) {
        $rule.info.antiAffineHostGroupName = $HostGroup
    }
    else {
        $rule.info.affineHostGroupName = $HostGroup
    }
    $spec.RulesSpec += $rule
    $cluster.ReconfigureComputeResource_Task($spec,$true) |
Out-Null
}

#########################
##   main processing   ##
#########################

#check if we need to display help and/or history
$HistoryText = @'
 Maintenance Log
 Date       By   Updates (newest updates at the top)
 ---------- ---- ---------------------------------------------
-----------------
 10/06/2015 sb   Initial release.
 06/21/2016 sb   Updated code to support refresh of existing
rules as well as
                 partial groups and rules creation.  Changed
default groups and
                 rule naming to simplify them.  Added the -
noruleupdate switch.
################################################################
#################
'@
$myvarScriptName = ".\add-DRSAffinityRulesForMA.ps1"
```

```
if ($help) {get-help $myvarScriptName; exit}
if ($History) {$HistoryText; exit}

#let's make sure PowerCLI is being used
if ((Get-PSSnapin VMware.VimAutomation.Core -ErrorAction
SilentlyContinue) -eq $null)#is it already there
{
          Add-PSSnapin VMware.VimAutomation.Core #no  let's add
it
          if (!$ ) #have we been able to add it successfully
          {
                    OutputLogData -category "ERROR" -message
"Unable to load the PowerCLI snapin.  Please make sure PowerCLI
is installed on this server."
                    return
          }
}

#let's load the Nutanix cmdlets
if ((Get-PSSnapin -Name NutanixCmdletsPSSnapin -ErrorAction
SilentlyContinue) -eq $null)#is it already there
{
          Add-PSSnapin NutanixCmdletsPSSnapin #no  let's add it
          if (!$ ) #have we been able to add it successfully
          {
                    OutputLogData -category "ERROR" -message
"Unable to load the Nutanix snapin.  Please make sure the
Nutanix Cmdlets are installed on this server."
                    return
          }
}

#initialize variables
          #misc variables
          $myvarElapsedTime =
[System.Diagnostics.Stopwatch]::StartNew() #used to store
script begin timestamp
          $myvarvCenterServers = @() #used to store the list of
all the vCenter servers we must connect to
          $myvarOutputLogFile = (Get-Date -UFormat
"%Y_%m_%d_%H_%M_")
          $myvarOutputLogFile += "OutputLog.log"

          ####################################################
######################
          # command line arguments initialization
          ####################################################
######################
          #let's initialize parameters if they haven't been
specified
          if (!$vcenter) {$vcenter = read-host "Enter vCenter
server name or IP address"}#prompt for vcenter server name
          $myvarvCenterServers = $vcenter.Split(",") #make sure
we parse the argument in case it contains several entries
          if (!$ntnx_cluster1) {$ntnx_cluster1 = read-host
"Enter the hostname or IP address of the first Nutanix
cluster"}#prompt for the first Nutanix cluster name
          if (!$ntnx_cluster2) {$ntnx_cluster2 = read-host
"Enter the hostname or IP address of the second Nutanix
cluster"}#prompt for the second Nutanix cluster name
          if (!$username) {$username = read-host "Enter the
Nutanix cluster username"}#prompt for the Nutanix cluster
username
          if (!$password) {$password = read-host "Enter the
Nutanix cluster password"}#prompt for the Nutanix cluster
password
     $spassword = $password | ConvertTo-SecureString -
AsPlainText -Force

          ##################################
          ##  Main execution here       ##
          ##################################

          #building a variable containing the Nutanix cluster
names
          $myvarNutanixClusters =
@($ntnx_cluster1,$ntnx_cluster2)
          #initialize variables we'll need to store information
about the Nutanix clusters
          $myvarNtnxC1_hosts, $myvarNtnxC2_hosts,
$myvarNtnxC1_MaActiveCtrs, $myvarNtnxC2_MaActiveCtrs = @()
          $myvarCounter = 1

          #connect to each Nutanix cluster to figure out the
info we need
          foreach ($myvarNutanixCluster in
$myvarNutanixClusters)
          {
                    OutputLogData -category "INFO" -message
"Connecting to the Nutanix cluster $myvarNutanixCluster..."
                    if (!($myvarNutanixClusterConnect = Connect-
NutanixCluster -Server $myvarNutanixCluster -UserName $username
-Password $spassword -AcceptInvalidSSLCerts -
ForcedConnection))#make sure we connect to the Nutanix cluster
OK...
                    {#error handling
```

```
                                $myvarerror =
$error[0].Exception.Message
                                OutputLogData -category "ERROR" -
message "$myvarerror"
                                break #exit since we can't connect
to one of the Nutanix clusters
                        }
                        else #...otherwise show confirmation
                        {
                                OutputLogData -category "INFO" -
message "Connected to Nutanix cluster $myvarNutanixCluster."
                        }#endelse

                        if ($myvarNutanixClusterConnect)
                        {


                ##########################################
                                # processing for each Nutanix
cluster here#
                ##########################################

                                if ($myvarCounter -eq 1)
                                {
                                        #retrieve hostnames of
nodes forming up this cluster
                                        OutputLogData -category
"INFO" -message "Getting hosts in $myvarNutanixCluster..."
                                        $myvarNtnxC1_hosts = get-
ntnxhost | Select -Property hypervisorAddress
                                        #retrieve container names
for active metro availability protection domains
                                        OutputLogData -category
"INFO" -message "Getting active metro availability protection
domains in $myvarNutanixCluster..."
                                        $myvarMaActivePDs = Get-
NTNXProtectionDomain | where {($_.active -eq $true) -and
($_.metroAvail.role -eq "Active")} #figure out which protection
domains are MA and active
                        $myvarNtnxC1_MaActiveCtrs = $myvarMaActivePDs |
%{$_.metroAvail.container}
                                }
                                if ($myvarCounter -eq 2)
                                {
                                        #retrieve hostnames of
nodes forming up this cluster
                                        OutputLogData -category
"INFO" -message "Getting hosts in $myvarNutanixCluster..."
                                        $myvarNtnxC2_hosts = get-
ntnxhost | Select -Property hypervisorAddress
                                        #retrieve container names
for active metro availability protection domains
                                        OutputLogData -category
"INFO" -message "Getting active metro availability protection
domains in $myvarNutanixCluster..."
                                        $myvarMaActivePDs = Get-
NTNXProtectionDomain | where {($_.active -eq $true) -and
($_.metroAvail.role -eq "Active")} #figure out which protection
domains are MA and active
                                        $myvarNtnxC2_MaActiveCtrs
= $myvarMaActivePDs | %{$_.metroAvail.container}
                                }

                        }#endif
                OutputLogData -category "INFO" -message
"Disconnecting from Nutanix cluster $myvarNutanixCluster..."
                        Disconnect-NutanixCluster -Servers
$myvarNutanixCluster #cleanup after ourselves and disconnect
from the Nutanix cluster

                        #increment the counter
                        ++$myvarCounter
                }#end foreach Nutanix cluster loop


                #connect to vcenter now
                foreach ($myvarvCenter in $myvarvCenterServers)
                {
                        OutputLogData -category "INFO" -message
"Connecting to vCenter server $myvarvCenter..."
                        if (!($myvarvCenterObject = Connect-VIServer
$myvarvCenter))#make sure we connect to the vcenter server
OK...
                        {#make sure we can connect to the vCenter
server
                                $myvarerror =
$error[0].Exception.Message
                                OutputLogData -category "ERROR" -
message "$myvarerror"
                                return
                        }
                        else #...otherwise show the error message
                        {
                                OutputLogData -category "INFO" -
message "Connected to vCenter server $myvarvCenter."
                        }#endelse
```

```
                if ($myvarvCenterObject)
                {

                        ###################################
                        #main processing for vcenter here#
                        ###################################


        ########################
        # PROCESS VMHOSTS

                        #let's match host IP addresses we
got from the Nutanix clusters to VMHost objects in vCenter
                        $myvarNtnxC1_vmhosts = @() #this is
where we will save the hostnames of the hosts which make up the
first Nutanix cluster
                        $myvarNtnxC2_vmhosts = @() #this is
where we will save the hostnames of the hosts which make up the
second Nutanix cluster
                        OutputLogData -category "INFO" -
message "Getting hosts registered in $myvarvCenter..."
                        $myvarVMHosts = Get-VMHost #get all
the vmhosts registered in vCenter
                        foreach ($myvarVMHost in
$myvarVMHosts) #let's look at each host and determine which is
which
                        {
                                OutputLogData -category
"INFO" -message "Retrieving vmk interfaces for $myvarVMHost..."
                                $myvarHostVmks =
$myvarVMHost.NetworkInfo.VirtualNic #retrieve all vmk NICs for
that host
                                foreach ($myvarHostVmk in
$myvarHostVmks) #examine all VMKs
                                {
                                        foreach
($myvarHostIP in $myvarNtnxC1_hosts) #compare to the host IP
addresses we got from the Nutanix cluster 1
                                        {
                                                if
($myvarHostVmk.IP -eq $myvarHostIP.hypervisorAddress)
                                                {

        OutputLogData -category "INFO" -message
"$myvarVMHost.Name is a host in $ntnx_cluster1..."

        $myvarNtnxC1_vmhosts += $myvarVMHost#if we get a
match, that vcenter host is in cluster 1
                                                }
                                        }#end foreacch
IP C1 loop
                                        foreach
($myvarHostIP in $myvarNtnxC2_hosts) #compare to the host IP
addresses we got from the Nutanix cluster 2
                                        {
                                                if
($myvarHostVmk.IP -eq $myvarHostIP.hypervisorAddress)
                                                {

        OutputLogData -category "INFO" -message
"$myvarVMHost.Name is a host in $ntnx_cluster2..."

        $myvarNtnxC2_vmhosts += $myvarVMHost #if we get a
match, that vcenter host is in cluster 2
                                                }
                                        }#end foreacch
IP C2 loop
                                }#end foreach VMK loop
                        }#end foreach VMhost loop

                        #check all vmhosts are part of the
same vSphere cluster
                        OutputLogData -category "INFO" -
message "Checking that all hosts are part of the same compute
cluster..."
                        $myvarvSphereCluster =
$myvarNtnxC1_vmhosts[0] | Get-Cluster  #we look at which
cluster the first vmhost in cluster 1 belongs to.
                        $myvarvSphereClusterName =
$myvarvSphereCluster.Name
                        $myvarvSphereClusterVMHosts =
$myvarNtnxC1_vmhosts + $myvarNtnxC2_vmhosts #let's create an
array with all vmhosts that should be in the compute cluster

                        #get existing DRS groups
                        $myvarDRSGroups = (get-cluster
$myvarvSphereClusterName).ExtensionData.ConfigurationEx.group

                        #get existing DRS rules
                        $myvarClusterComputeResourceView = Get-View -
ViewType ClusterComputeResource -Property Name, ConfigurationEx
| where-object {$_.Name -eq $myvarvSphereClusterName}
                        $myvarClusterDRSRules =
$myvarClusterComputeResourceView.ConfigurationEx.Rule

                        foreach ($myvarvSphereClusterVMHost in
$myvarvSphereClusterVMHosts) #let's now lok at each vmhost and
which cluster they belong to
```

```
                                {
                                        $myvarVMHostCluster =
$myvarvSphereClusterVMHost | Get-Cluster #which cluster does
this host belong to
                                        if ($myvarVMHostCluster -
ne $myvarvSphereCluster) #let's check if it's the same cluster
as our first host
                                        {
                                                $myvarVMHostName
= $myvarvSphereClusterVMHost.Name

          $myvarVMHostClusterName = $myvarVMHostCluster.Name
                                                OutputLogData -
category "ERROR" -message "$myvarVMHostName belongs to vSphere
cluster $myvarVMHostClusterName when it should be in
$myvarvSphereClusterName..."
                                                break #we'l stop
right here since at least one vmhost is not in the right
compute cluster
                                        }
                                }#end foreach cluster vmhost loop

                                #check that vSphere cluster has HA
and DRS enabled
                                OutputLogData -category "INFO" -
message "Checking HA is enabled on $myvarvSphereClusterName..."
                                if ($myvarvSphereCluster.HaEnabled
-ne $true) {OutputLogData -category "WARN" -message "HA is not
enabled on $myvarvSphereClusterName!"}
                                OutputLogData -category "INFO" -
message "Checking DRS is enabled on
$myvarvSphereClusterName..."
                                if ($myvarvSphereCluster.DrsEnabled
-ne $true)
                                {
                                        OutputLogData -category
"ERROR" -message "DRS is not enabled on
$myvarvSphereClusterName!"
                                        break #exit since DRS is
not enabled
                                }

            #check to see if the host group already exists
            $myvarDRSHostGroups = $myvarDRSGroups | {$_.host}
#keep host groups

            #CREATE DRS affinity groups for hosts in each
nutanix cluster
            $myvarNtnxC1_DRSHostGroupName = "DRS_HG_MA_" +
$ntnx_cluster1
            $myvarNtnxC2_DRSHostGroupName = "DRS_HG_MA_" +
$ntnx_cluster2

            #do we have an existing DRS host group for c1
already
            if ($myvarDRSHostGroups | Where-Object {$_.Name -eq
$myvarNtnxC1_DRSHostGroupName})
            { #yes, so let's update it
                OutputLogData -category "INFO" -message
"Updating DRS Host Group $myvarNtnxC1_DRSHostGroupName on
cluster $myvarvSphereCluster"
                Update-DrsHostGroup -cluster
$myvarvSphereCluster -Hosts $myvarNtnxC1_vmhosts -groupHostName
$myvarNtnxC1_DRSHostGroupName
            }
            else
            { #no, so let's create it
                OutputLogData -category "INFO" -message
"Creating DRS Host Group $myvarNtnxC1_DRSHostGroupName on
cluster $myvarvSphereClusterName for $ntnx_cluster1..."
                                $myvarNtnxC1_vmhosts | New-
DrsHostGroup -Name $myvarNtnxC1_DRSHostGroupName -Cluster
$myvarvSphereCluster
            }

            #do we have an existing DRS host group for c2
already
            if ($myvarDRSHostGroups | Where-Object {$_.Name -eq
$myvarNtnxC2_DRSHostGroupName})
            { #yes, so let's update it
                OutputLogData -category "INFO" -message
"Updating DRS Host Group $myvarNtnxC2_DRSHostGroupName on
cluster $myvarvSphereCluster"
                Update-DrsHostGroup -cluster
$myvarvSphereCluster -Hosts $myvarNtnxC2_vmhosts -groupHostName
$myvarNtnxC2_DRSHostGroupName
            }
            else
            { #no, so let's create it
                OutputLogData -category "INFO" -message
"Creating DRS Host Group $myvarNtnxC2_DRSHostGroupName on
cluster $myvarvSphereClusterName for $ntnx_cluster2..."
                                $myvarNtnxC2_vmhosts | New-
DrsHostGroup -Name $myvarNtnxC2_DRSHostGroupName -Cluster
$myvarvSphereCluster
            }
```

```
                        ######################
            # PROCESS VMS and RULES

            #check existing vm groups
            $myvarDRSVMGroups = $myvarDRSGroups | {$_.vm} #keep
vm groups

            #retrieve names of VMs in each active datastore
                        $myvarNtnxC1_vms, $myvarNtnxC2_vms
= @()


############################################################
            #Process VM DRS Groups and DRS Rules for Nutanix
cluster 1
            foreach ($myvarDatastore in
$myvarNtnxC1_MaActiveCtrs)
                        {
                                    OutputLogData -category
"INFO" -message "Getting VMs in datastore $myvarDatastore..."
                                    $myvarNtnxC1_vms += Get-
Datastore -Name $myvarDatastore | Get-VM

                $myvarDRSVMGroupName = "DRS_VM_MA_" +
$myvarDatastore


                #compare. if not exist then create
                if (!($myvarDRSVMGroups | Where-Object {$_.Name
-eq $myvarDRSVMGroupName})) #the DRS VM Group does not exist,
so let's create it
                {
                                    OutputLogData -
category "INFO" -message "Creating DRS VM Group
$myvarDRSVMGroupName on cluster $myvarvSphereClusterName for
datastore $myvarDatastore which is active on $ntnx_cluster1..."
                                    $myvarNtnxC1_vms |
New-DrsVMGroup -Name $myvarDRSVMGroupName -Cluster
$myvarvSphereCluster
                                    }
                else
                {
                    #else edit existing
                    OutputLogData -category "INFO" -message
"Updating DRS VM Group $myvarDRSVMGroupName on cluster
$myvarvSphereClusterName for datastore $myvarDatastore which is
active on $ntnx_cluster1..."
                                    Update-DrsVMGroup -cluster
$myvarvSphereCluster -VMs $myvarNtnxC1_vms -groupVMName
$myvarDRSVMGroupName
                }


                #retrieve DRS rule
                $myvarDRSRuleName = "DRS_Rule_MA_" +
$myvarDatastore


                #if not exist create
                if (!($myvarClusterDRSRules | Where-Object
{$_.Name -eq $myvarDRSRuleName})) #the DRS VM Group does not
exist, so let's create it
                {
                                                #create DRS affinity
rules for VMs to Hosts
                                                OutputLogData -
category "INFO" -message "Creating DRS rule $myvarDRSRuleName
on cluster $myvarvSphereCluster so that VMs in
$myvarDRSVMGroupName should run on hosts in
$myvarNtnxC1_DRSHostGroupName..."

                                                New-DrsVMToHostRule -
VMGroup $myvarDRSVMGroupName -HostGroup
$myvarNtnxC1_DRSHostGroupName -Name $myvarDRSRuleName -Cluster
$myvarvSphereCluster
                }
                else #the DRS rule is already there
                {

                    if (!($noruleupdate))
                    {
                        OutputLogData -category "INFO" -message
"Updating DRS rule $myvarDRSRuleName on cluster
$myvarvSphereCluster for $myvarDatastore..."
                        Update-DRSVMToHostRule -VMGroup
$myvarDRSVMGroupName -HostGroup $myvarNtnxC1_DRSHostGroupName -
Name $myvarDRSRuleName -Cluster $myvarvSphereCluster -RuleKey
$(($myvarClusterDRSRules | Where-Object {$_.Name -eq
$myvarDRSRuleName}).Key) -RuleUuid $(($myvarClusterDRSRules |
Where-Object {$_.Name -eq $myvarDRSRuleName}).RuleUuid)
                    }
                }


                                    }#end foreach datastore in C1 loop
```

```
#########################################################
            #Process VM DRS Groups and DRS Rules for Nutanix
cluster 2
                              foreach ($myvarDatastore in
$myvarNtnxC2_MaActiveCtrs)
                              {
                                        OutputLogData -category
"INFO" -message "Getting VMs in datastore $myvarDatastore..."
                                        $myvarNtnxC2_vms += Get-
Datastore -Name $myvarDatastore | Get-VM

                $myvarDRSVMGroupName = "DRS_VM_MA_" +
$myvarDatastore


                #compare. if not exist then create
                if (!($myvarDRSVMGroups | Where-Object {$_.Name
-eq $myvarDRSVMGroupName}))
                {
                    OutputLogData -category "INFO" -message
"Creating DRS VM Group $myvarDRSVMGroupName on cluster
$myvarvSphereClusterName for datastore $myvarDatastore which is
active on $ntnx_cluster2..."
                                        $myvarNtnxC2_vms |
New-DrsVMGroup -Name $myvarDRSVMGroupName -Cluster
$myvarvSphereCluster
                                    }
                else #else edit existing
                {
                    OutputLogData -category "INFO" -message
"Updating DRS VM Group $myvarDRSVMGroupName on cluster
$myvarvSphereClusterName for datastore $myvarDatastore which is
active on $ntnx_cluster2..."
                    Update-DrsVMGroup -cluster
$myvarvSphereCluster -VMs $myvarNtnxC2_vms -groupVMName
$myvarDRSVMGroupName
                }


                $myvarDRSRuleName = "DRS_Rule_MA_" +
$myvarDatastore
                #retrieve DRS rule
                #if not exist create
                if (!($myvarClusterDRSRules | Where-Object
{$_.Name -eq $myvarDRSRuleName}))
                {
                                        #create DRS affinity
rules for VMs to Hosts
                                        OutputLogData -
category "INFO" -message "Creating DRS rule
$myvarDRSVMGroupName on cluster $myvarvSphereClusterName so
that VMs in $myvarDRSVMGroupName should run on hosts in
$myvarNtnxC2_DRSHostGroupName..."

                                        New-DrsVMToHostRule -
VMGroup $myvarDRSVMGroupName -HostGroup
$myvarNtnxC2_DRSHostGroupName -Name $myvarDRSRuleName -Cluster
$myvarvSphereCluster
                }
                else #the DRS rule is already there
                {

                    if (!($noruleupdate))
                    {
                        OutputLogData -category "INFO" -message
"Updating DRS rule $myvarDRSVMGroupName on cluster
$myvarvSphereClusterName for $myvarDatastore..."
                        Update-DRSVMToHostRule -VMGroup
$myvarDRSVMGroupName -HostGroup $myvarNtnxC2_DRSHostGroupName -
Name $myvarDRSRuleName -Cluster $myvarvSphereCluster -RuleKey
$(($myvarClusterDRSRules | Where-Object {$_.Name -eq
$myvarDRSRuleName}).Key) -RuleUuid $(($myvarClusterDRSRules |
Where-Object {$_.Name -eq $myvarDRSRuleName}).RuleUuid)
                    }
                }

                            }#end foreach datastore in C2 loop



            }#endif
        OutputLogData -category "INFO" -message "Disconnecting
from vCenter server $vcenter..."
                Disconnect-viserver -Confirm:$False #cleanup
after ourselves and disconnect from vcenter
          }#end foreach vCenter


#########################
##       cleanup       ##
#########################

          #let's figure out how much time this all took
          OutputLogData -category "SUM" -message "total
processing time: $($myvarElapsedTime.Elapsed.ToString())"
```

```
          #cleanup after ourselves and delete all custom
variables
          Remove-Variable myvar*
          Remove-Variable ErrorActionPreference
          Remove-Variable help
          Remove-Variable history
          Remove-Variable log
          Remove-Variable ntnx_cluster1
          Remove-Variable ntnx_cluster2
          Remove-Variable username
          Remove-Variable password
          Remove-Variable vcenter
          Remove-Variable debugme
```

Historique des Versions

Version	Date	Editeur	Description
1.0	11 Juillet 2016	Stéphane Bourdeaud (stephane.bourdeaud@nutanix.com)	Création du document

La dernière version de ce document est disponible dans la base documentaire de Nutanix Services Consulting.