Work Instructions for:

Metro Availability on VMware vSphere

Table of Contents

Executive Summary

This document contains specific work instructions describing how to use Nutanix Metro Availability features on VMware vSphere. It is meant to be used by operational teams who manage a Nutanix environment.

Specifically, this document describes the following two procedures:

1. How to perform a planned failover for a Metro Availability (MA) protection domain (PD): this procedure is meant to be used when you need to do a planned maintenance on one site and you need all your workloads to run on the other site. Starting with Acropolis OS (AOS) v4.6, this can be done without service interruption for your virtual machines. The procedure also explains how to go back to the original site (failback).
2. How to recover an MA PD after a site failure has occurred: this procedure is meant to be used when a disaster or failure has occurred and you need to recover your virtual machines on the remaining site. The procedure also explains what to do after the site has been recovered (failback).

Stop

Note that some of the actions described in these procedures are destructive, meaning that if they are not performed correctly, they can overwrite your data.
Use extreme care especially before doing any replication re-enablement.

Procedure #1: How to perform a planned failover and failback for a Metro Availability protection domain

Overview

You perform a planned failover when you need to move everything (your virtual machines and your active storage) to a single site. For example, assuming you have two datacenters (dc1 and dc2), you need to take dc1 offline for maintenance and therefore you want to move everything over to dc2 without causing a service interruption.

When you are done with the maintenance and dc1 comes back online, you will want to move everything back. This is what we call the failback procedure.

The failover process is described using precise terminology. The site we are moving from is called the source site (in our example, the source site is called dc1). The site we are moving to is the target site (in our example, the target site is called dc2).

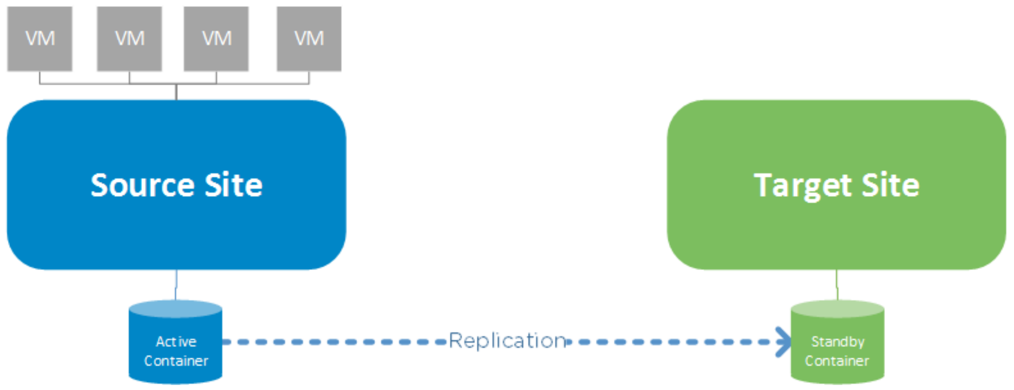
For the failback process, we are moving from the target site (exp: dc2) back to the source site (exp: dc1). We do not change site denomination in order to avoid confusion, but you need to be extremely careful about which site is source and which site is target.

Note that for each process, there is:

1. An initial situation: this shows which site is source, which is target, where VMs are running, where active storage/container is and which direction the replication goes.
2. A process workflow: showing the major steps of the process. Destructive actions are in orange and display a warning logo. When performing those actions, you must make sure your VMs and their data is in the right site, or you may end up overwriting active data with stale data.
3. A resulting situation: this shows how things look after the procedure has been performed.

Planned Failover

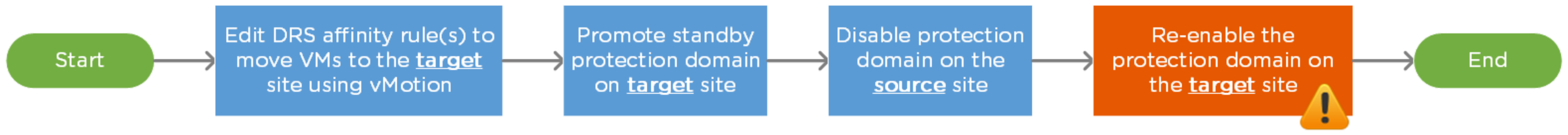
This is the initial situation:



Note that:

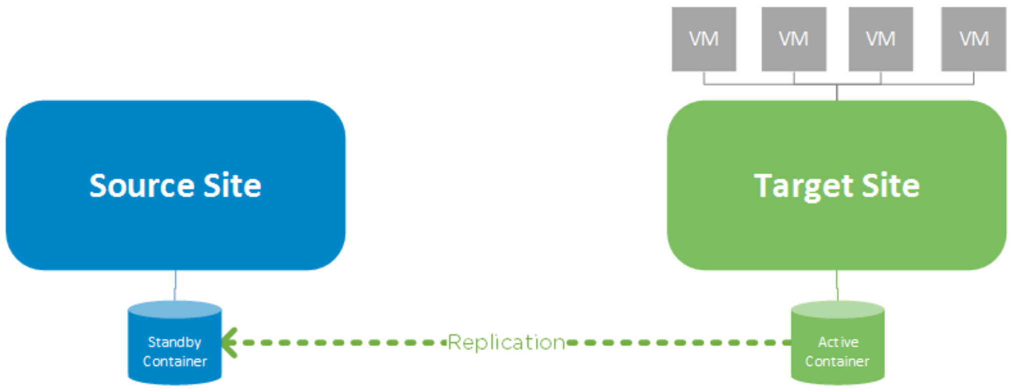
1. VMs are running on the source site (dc1) where storage is active.
2. Replication is going from the source site (dc1) to the target site (dc2)

The process workflow is as follows:


Start
Edit DRS affinity rule(s) to move VMs to the target site using vMotion
Promote standby protection domain on target site
Disable protection domain on the source site
Re-enable the protection domain on the target site
End

Detailed steps along with screenshots are included below. This is just an overview workflow to be used as a reference.

The resulting situation will be as follows:



Note that:

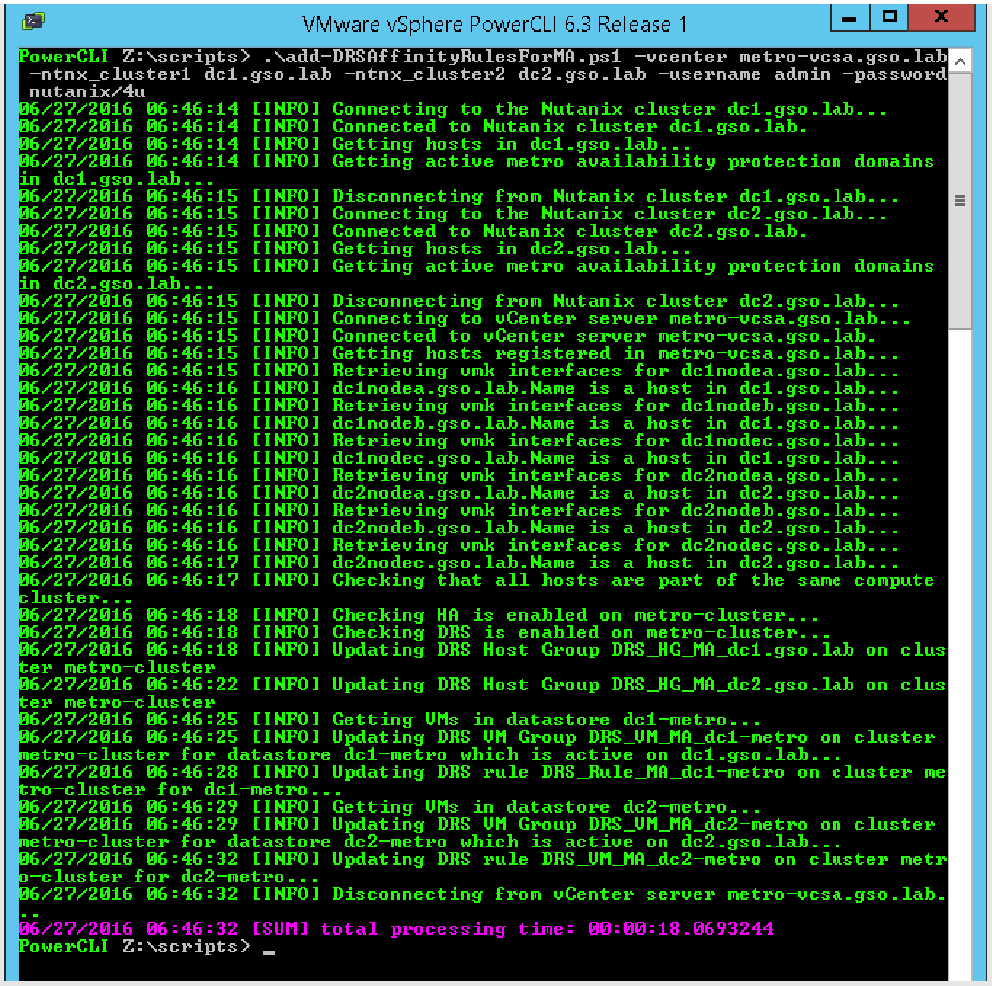
1. VMs will run on the target site (dc2)
2. Storage will be active on the target site (dc2)
3. Replication will run from dc2 to dc1

In the detailed steps example below, note that:

1. The **source** site is called **dc1**.
2. The **target** site is called **dc2**.
3. The **container** is called **dc1-metro**.
4. The **protection domain** is also called **dc1-metro**.
5. VMs vm1 to vm10 normally run on dc1.
6. The compute cluster is called metro-cluster.

7. Some of the screenshots show other containers and protection domains. This is because dc2 is also replicated to dc1, which is most likely also the case of your production environment.

1.



```
PowerCLI Z:\scripts> .\add-DRSAffinityRulesForMA.ps1 -vcenter metro-vcsa.gso.lab
 -ntnx_cluster1 dc1.gso.lab -ntnx_cluster2 dc2.gso.lab -username admin -password
 nutanix/4u
06/27/2016 06:46:14 [INFO] Connecting to the Nutanix cluster dc1.gso.lab...
06/27/2016 06:46:14 [INFO] Connected to Nutanix cluster dc1.gso.lab.
06/27/2016 06:46:14 [INFO] Getting hosts in dc1.gso.lab...
06/27/2016 06:46:14 [INFO] Getting active metro availability protection domains
in dc1.gso.lab...
06/27/2016 06:46:15 [INFO] Disconnecting from Nutanix cluster dc1.gso.lab...
06/27/2016 06:46:15 [INFO] Connecting to the Nutanix cluster dc2.gso.lab...
06/27/2016 06:46:15 [INFO] Connected to Nutanix cluster dc2.gso.lab.
06/27/2016 06:46:15 [INFO] Getting hosts in dc2.gso.lab...
06/27/2016 06:46:15 [INFO] Getting active metro availability protection domains
in dc2.gso.lab...
06/27/2016 06:46:15 [INFO] Disconnecting from Nutanix cluster dc2.gso.lab...
06/27/2016 06:46:15 [INFO] Connecting to vCenter server metro-vcsa.gso.lab...
06/27/2016 06:46:15 [INFO] Connected to vCenter server metro-vcsa.gso.lab.
06/27/2016 06:46:15 [INFO] Getting hosts registered in metro-vcsa.gso.lab...
06/27/2016 06:46:15 [INFO] Retrieving vmk interfaces for dc1nodea.gso.lab...
06/27/2016 06:46:16 [INFO] dc1nodea.gso.lab.Name is a host in dc1.gso.lab...
06/27/2016 06:46:16 [INFO] Retrieving vmk interfaces for dc1nodeb.gso.lab...
06/27/2016 06:46:16 [INFO] dc1nodeb.gso.lab.Name is a host in dc1.gso.lab...
06/27/2016 06:46:16 [INFO] Retrieving vmk interfaces for dc1nodec.gso.lab...
06/27/2016 06:46:16 [INFO] dc1nodec.gso.lab.Name is a host in dc1.gso.lab...
06/27/2016 06:46:16 [INFO] Retrieving vmk interfaces for dc2nodea.gso.lab...
06/27/2016 06:46:16 [INFO] dc2nodea.gso.lab.Name is a host in dc2.gso.lab...
06/27/2016 06:46:16 [INFO] Retrieving vmk interfaces for dc2nodeb.gso.lab...
06/27/2016 06:46:16 [INFO] dc2nodeb.gso.lab.Name is a host in dc2.gso.lab...
06/27/2016 06:46:16 [INFO] Retrieving vmk interfaces for dc2nodec.gso.lab...
06/27/2016 06:46:17 [INFO] dc2nodec.gso.lab.Name is a host in dc2.gso.lab...
06/27/2016 06:46:17 [INFO] Checking that all hosts are part of the same compute
cluster...
06/27/2016 06:46:18 [INFO] Checking HA is enabled on metro-cluster...
06/27/2016 06:46:18 [INFO] Checking DRS is enabled on metro-cluster...
06/27/2016 06:46:18 [INFO] Updating DRS Host Group DRS_HG_MA_dc1.gso.lab on clus
ter metro-cluster
06/27/2016 06:46:22 [INFO] Updating DRS Host Group DRS_HG_MA_dc2.gso.lab on clus
ter metro-cluster
06/27/2016 06:46:25 [INFO] Getting VMs in datastore dc1-metro...
06/27/2016 06:46:25 [INFO] Updating DRS VM Group DRS_VM_MA_dc1-metro on cluster
metro-cluster for datastore dc1-metro which is active on dc1.gso.lab...
06/27/2016 06:46:28 [INFO] Updating DRS rule DRS_Rule_MA_dc1-metro on cluster me
tro-cluster for dc1-metro...
06/27/2016 06:46:29 [INFO] Getting VMs in datastore dc2-metro...
06/27/2016 06:46:29 [INFO] Updating DRS VM Group DRS_VM_MA_dc2-metro on cluster
metro-cluster for datastore dc2-metro which is active on dc2.gso.lab...
06/27/2016 06:46:32 [INFO] Updating DRS rule DRS_VM_MA_dc2-metro on cluster metr
o-cluster for dc2-metro...
06/27/2016 06:46:32 [INFO] Disconnecting from vCenter server metro-vcsa.gso.lab.
..
06/27/2016 06:46:32 [SUM] total processing time: 00:00:18.0693244
PowerCLI Z:\scripts> _
```

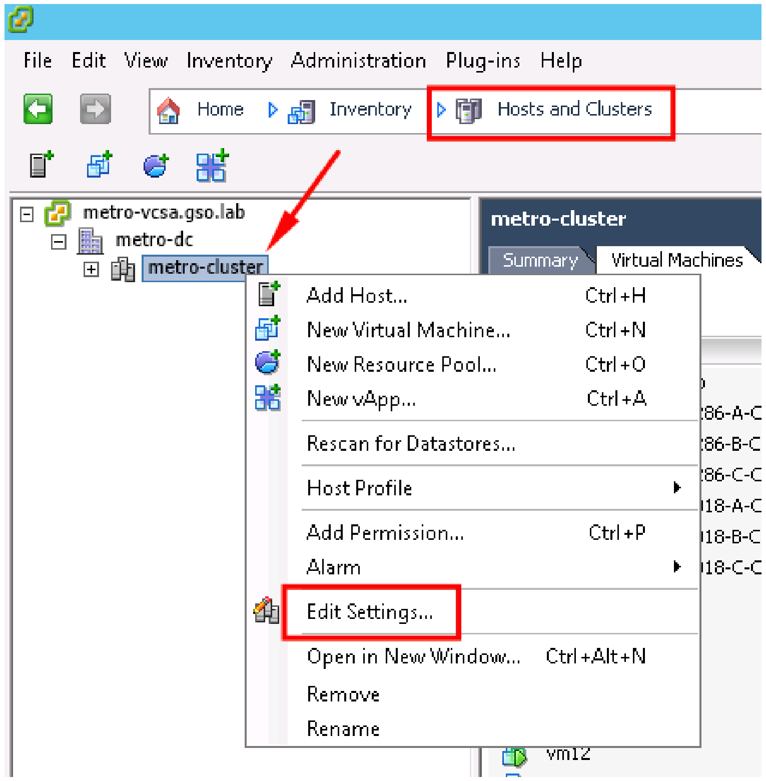
The first step is to make sure our DRS affinity rules are up to date and that VMs are where they belong.
This is very important because when we disable replication, storage will be active on both sites. If VMs are also running on both sites, our storage will be decoupled and when we re-enable replication, we will overwrite active data.

In order to update our DRS affinity rules, we use the **add-DRSAffinityRulesForMA.ps1** script which can be found in the Appendix of this document.

Note

The script will create the groups and rules if they do not exist already, and it will update them is they are already there.

2.



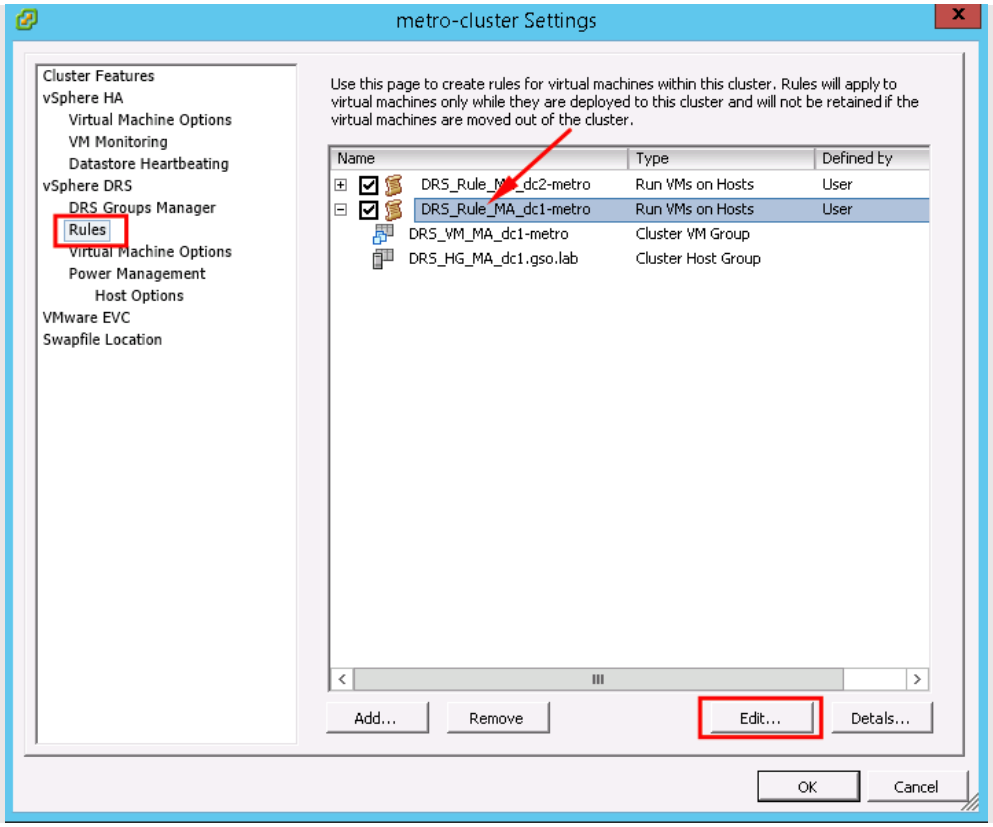
Stop

Before doing anything else, make sure in Prism that your *dc1-metro* protection domain is **Active** on dc1 and **Standby** on dc2 and that the replication status is **Enabled (in sync)**.

Now that our DRS affinity rules are up to date, we edit the *dc1-metro* rule to change VMs affinity from dc1 hosts to dc2 hosts.

Start by right clicking the cluster in the vSphere client and select **Edit Settings**.

3.



metro-cluster Settings

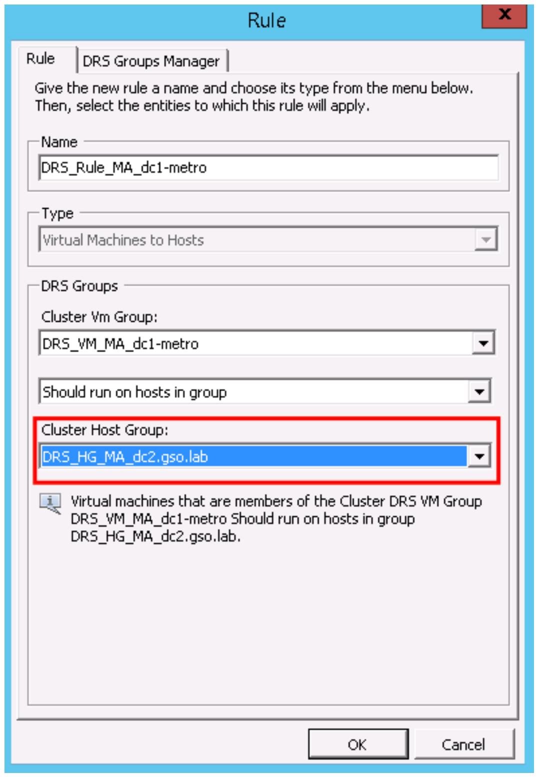
Cluster Features
vSphere HA
Virtual Machine Options
VM Monitoring
Datastore Heartbeating
vSphere DRS
DRS Groups Manager
Rules
Virtual Machine Options
Power Management
Host Options
VMware EVC
Swapfile Location

Use this page to create rules for virtual machines within this cluster. Rules will apply to virtual machines only while they are deployed to this cluster and will not be retained if the virtual machines are moved out of the cluster.

Name	Type	Defined by
DRS_Rule_MA_dc2-metro	Run VMs on Hosts	User
DRS_Rule_MA_dc1-metro	Run VMs on Hosts	User
DRS_VM_MA_dc1-metro	Cluster VM Group	
DRS_HG_MA_dc1.gso.lab	Cluster Host Group	

Select **Rules** on the left pane, then select the rule for dc1-metro and click on **Edit** in the bottom right corner.

4.



Give the new rule a name and choose its type from the menu below.
Then, select the entities to which this rule will apply.

Name: DRS_Rule_MA_dc1-metro

Type: Virtual Machines to Hosts

DRS Groups

Cluster Vm Group: DRS_VM_MA_dc1-metro

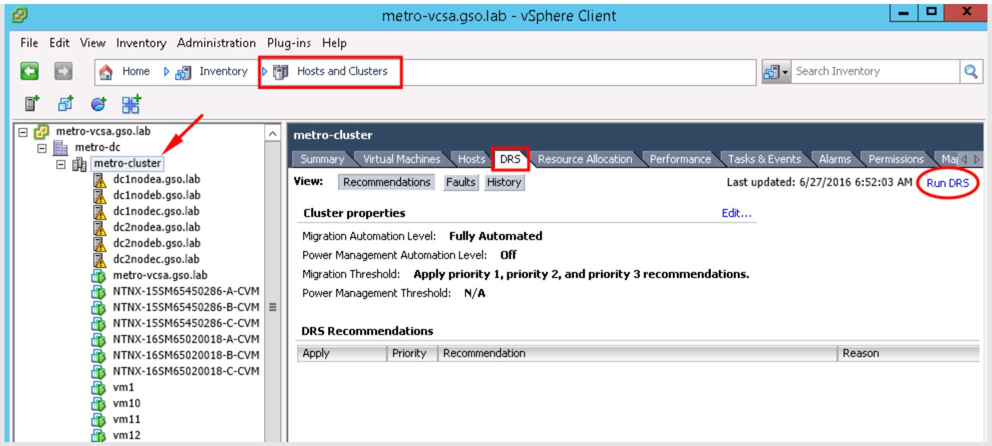
Should run on hosts in group

Cluster Host Group: DRS_HG_MA_dc2.gso.lab

Virtual machines that are members of the Cluster DRS VM Group DRS_VM_MA_dc1-metro Should run on hosts in group DRS_HG_MA_dc2.gso.lab.

Change the **Cluster Host Group** to select the dc2 host group, then click **OK**.

5.



metro-vcsa.gso.lab - vSphere Client

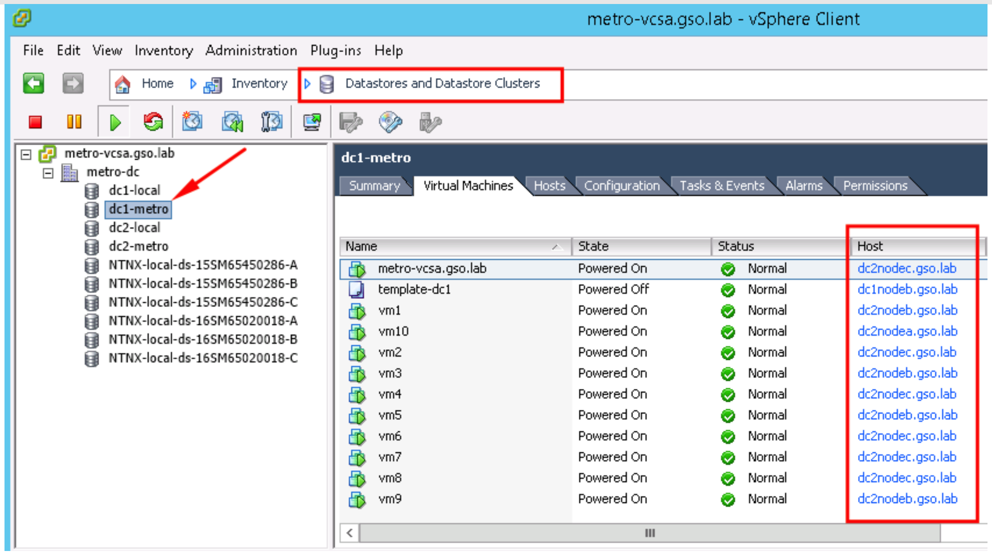
Migration Automation Level: **Fully Automated**
Power Management Automation Level: **Off**
Migration Threshold: **Apply priority 1, priority 2, and priority 3 recommendations.**
Power Management Threshold: **N/A**

Last updated: 6/27/2016 6:52:03 AM

Apply	Priority	Recommendation	Reason

Once the affinity rule has been edited, we can force DRS to apply the new rules, at which point virtual machines will be moved using vMotion to hosts on dc2.

6.



metro-vcsa.gso.lab - vSphere Client

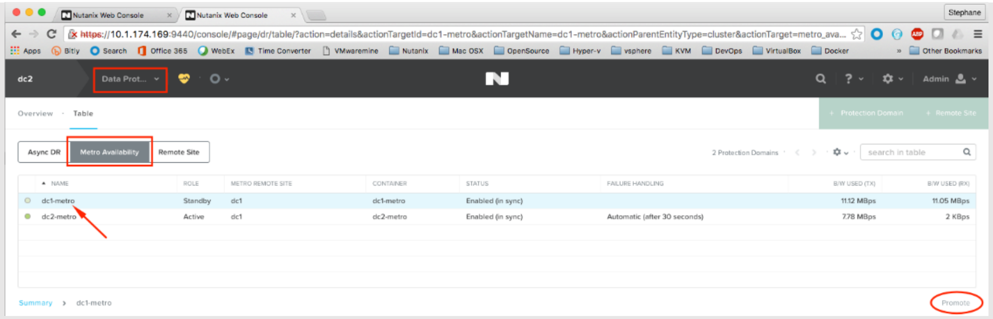
Name	State	Status	Host
metro-vcsa.gso.lab	Powered On	Normal	dc2nodec.gso.lab
template-dc1	Powered Off	Normal	dc1nodeb.gso.lab
vm1	Powered On	Normal	dc2nodeb.gso.lab
vm10	Powered On	Normal	dc2nodea.gso.lab
vm2	Powered On	Normal	dc2nodec.gso.lab
vm3	Powered On	Normal	dc2nodeb.gso.lab
vm4	Powered On	Normal	dc2nodec.gso.lab
vm5	Powered On	Normal	dc2nodeb.gso.lab
vm6	Powered On	Normal	dc2nodec.gso.lab
vm7	Powered On	Normal	dc2nodec.gso.lab
vm8	Powered On	Normal	dc2nodec.gso.lab
vm9	Powered On	Normal	dc2nodeb.gso.lab

We can verify that all VMs in the *dc1-metro* container are now running on dc2 hosts by selecting the *dc1-metro* datastore in the vSphere client.

If for any reason some VMs have not been moved by DRS, you will need to migrate them manually to a host in dc2.

Note that templates will not move, but this is fine since templates do not actively change their data.

7.



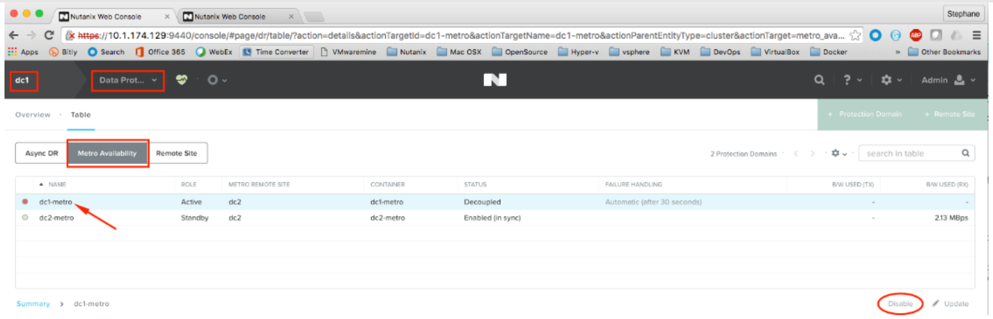
NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Standby	dc1	dc1-metro	Enabled (in sync)		11.12 MBps	11.05 MBps
dc2-metro	Active	dc1	dc2-metro	Enabled (in sync)	Automatic (after 30 seconds)	7.78 MBps	2 KBps

Now that we are positive our VMs are all running on dc2 hosts, we can promote the *dc1-metro* protection domain on the target site (dc2) from dc2 Prism interface.
Note that Prism will ask you to confirm by having you type **Promote**.

Note

If you want to minimize the amount of data to replicate when we will re-enable replication later, you can also manually take a snapshot for the protection domain before doing the promotion. This should be done from the source site.

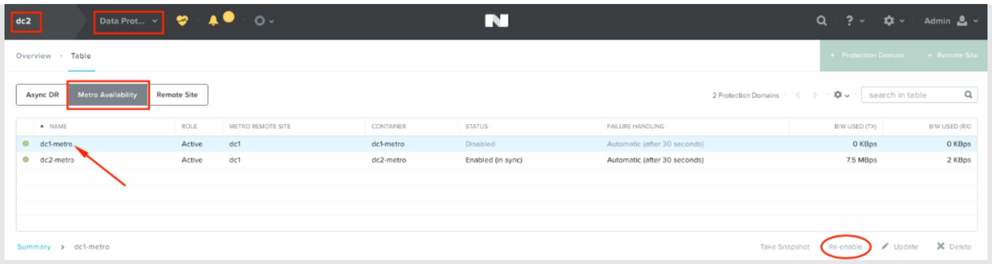
8.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Active	dc2	dc1-metro	Decoupled	Automatic (after 30 seconds)	-	-
dc2-metro	Standby	dc2	dc2-metro	Enabled (in sync)		-	2.13 MBps

Once the protection domain has been promoted on the target site (dc2), we must disable it on the source site (dc1).

9.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Active	dc1	dc1-metro	Disabled	Automatic (after 30 seconds)	0 KBps	0 KBps
dc2-metro	Active	dc1	dc2-metro	Enabled (in sync)	Automatic (after 30 seconds)	7.5 MBps	2 KBps

Stop

The action described here will overwrite data on dc1 (the source site). You must be absolutely sure that there are no active VMs on dc1 in the *dc1-metro* container before re-enabling replication. If there are active VMs, use storage vMotion to move them to another datastore first.

We can now re-enable replication from the target site (dc2) to the source site (dc1).

Note

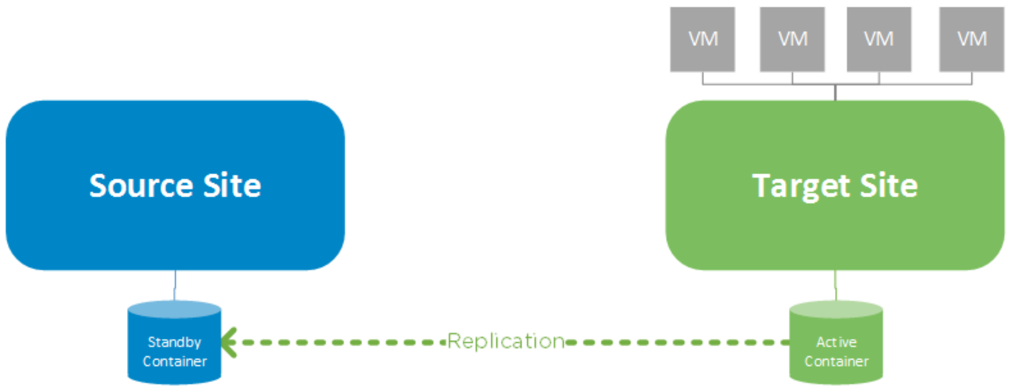
Even if we later take our source site (dc1) offline for maintenance, it is better to keep the protection domain replicated until this happens. This will also ensure that replication resumes as soon as dc1 comes back online after maintenance.

We are done with the planned failover procedure!

Planned Failback

Use this procedure after you have performed a planned failover.

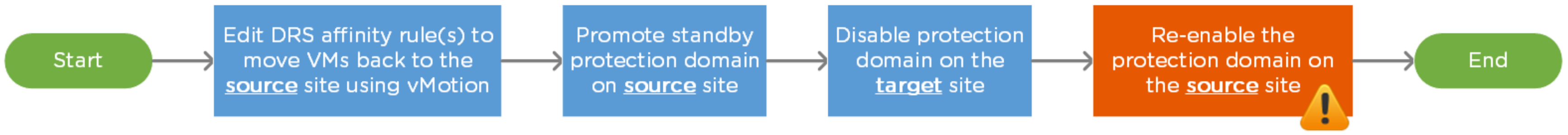
This is the initial situation:



Note that:

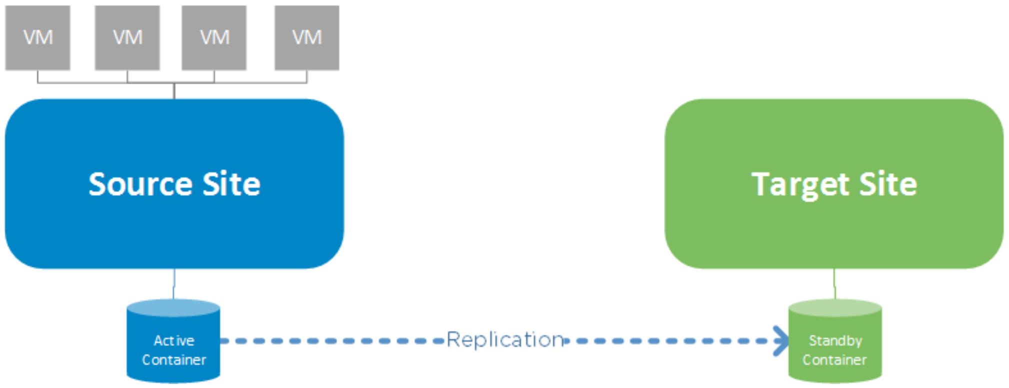
1. VMs are running on the target site (dc2) where storage is active.
2. Replication is going from the target site (dc2) to the source site (dc1)

The process workflow is as follows:


Start
Edit DRS affinity rule(s) to move VMs back to the source site using vMotion
Promote standby protection domain on source site
Disable protection domain on the target site
Re-enable the protection domain on the source site
End

Detailed steps along with screenshots are included below. This is just an overview workflow to be used as a reference.

The resulting situation will be as follows:



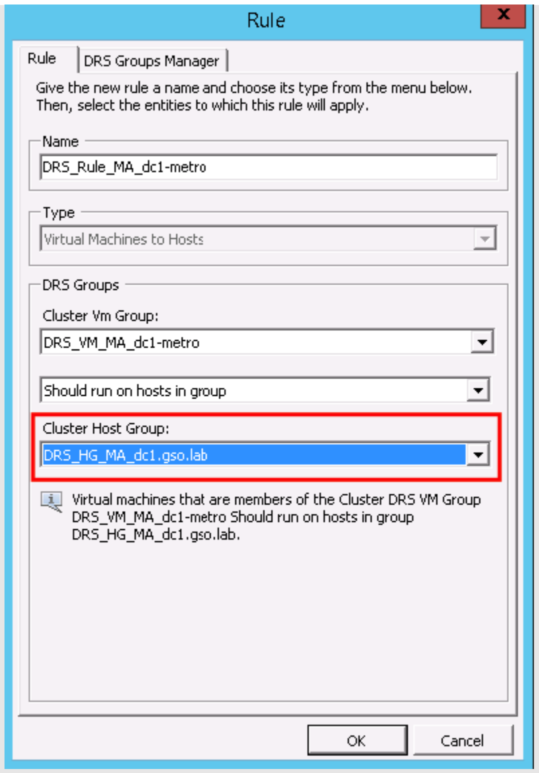
Note that:

1. VMs will run on the source site (dc1)
2. Storage will be active on the source site (dc1)
3. Replication will run from dc1 to dc2

In the detailed steps example below, note that:

1. The source site is called dc1.
2. The target site is called dc2.
3. The container is called dc1-metro.
4. The protection domain is also called dc1-metro.
5. VMs vm1 to vm10 normally run on dc1.
6. The compute cluster is called metro-cluster.

1.



Rule

Give the new rule a name and choose its type from the menu below. Then, select the entities to which this rule will apply.

Name: DRS_Rule_MA_dc1-metro

Type: Virtual Machines to Hosts

DRS Groups

Cluster Vm Group: DRS_VM_MA_dc1-metro

Should run on hosts in group

Cluster Host Group: DRS_HG_MA_dc1.gso.lab

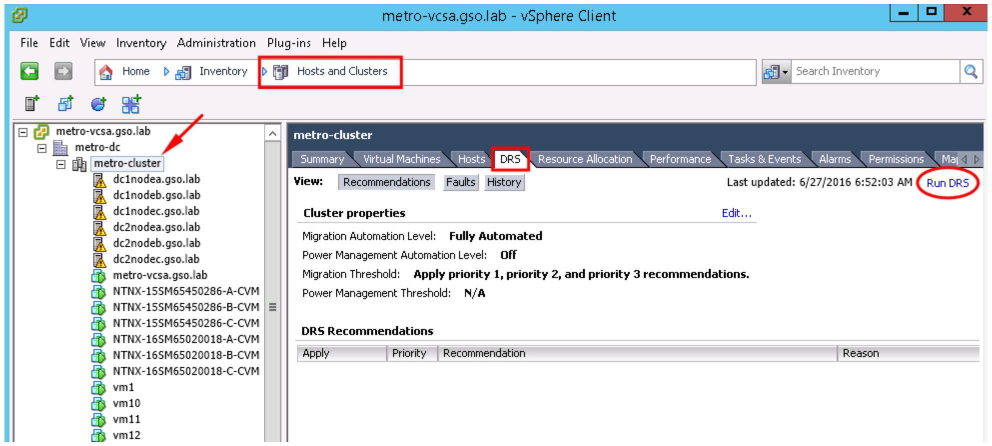
Virtual machines that are members of the Cluster DRS VM Group DRS_VM_MA_dc1-metro Should run on hosts in group DRS_HG_MA_dc1.gso.lab.

Stop

Before doing anything else make sure in Prism that the *dc1-metro* protection domain replication status is **Enabled (in sync)**.

We start by editing the *dc1-metro* DRS affinity rule and selecting the dc1 host group so that DRS will migrate virtual machines back to dc1 using vMotion.

2.


metro-vcsa.gso.lab - vSphere Client
File Edit View Inventory Administration Plug-ins Help
Home Inventory Hosts and Clusters
Search Inventory
Summary Virtual Machines Hosts DRS Resource Allocation Performance Tasks & Events Alarms Permissions
Run DRS
Edit...

- metro-vcsa.gso.lab
  - metro-dc
    - metro-cluster
      - dc1nodea.gso.lab
      - dc1nodeb.gso.lab
      - dc1nodec.gso.lab
      - dc2nodea.gso.lab
      - dc2nodeb.gso.lab
      - dc2nodec.gso.lab
      - metro-vcsa.gso.lab
      - NTNX-15SM65450286-A-CVM
      - NTNX-15SM65450286-B-CVM
      - NTNX-15SM65450286-C-CVM
      - NTNX-16SM65020018-A-CVM
      - NTNX-16SM65020018-B-CVM
      - NTNX-16SM65020018-C-CVM
      - vm1
      - vm10
      - vm11
      - vm12

metro-cluster

View: Recommendations Faults History

Last updated: 6/27/2016 6:52:03 AM

Cluster properties

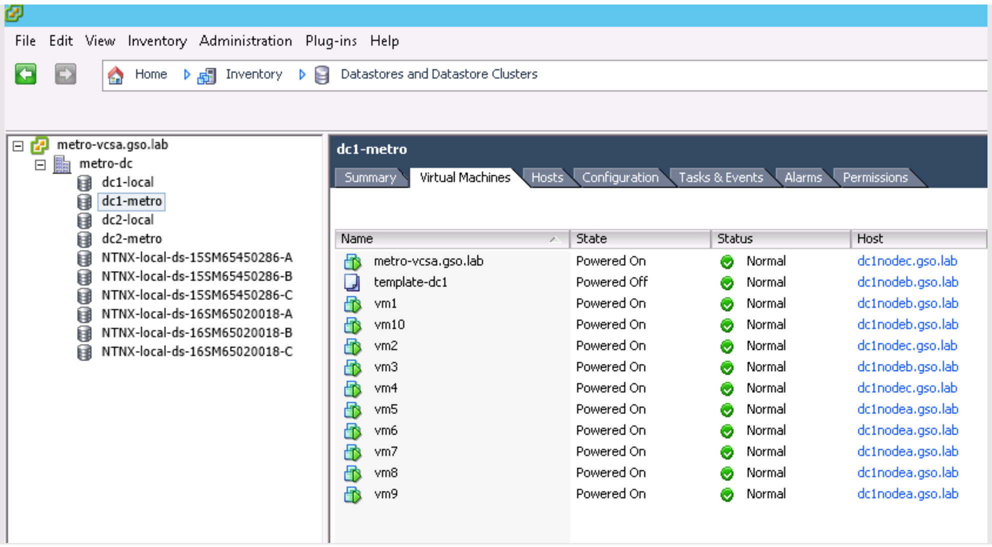
Migration Automation Level: **Fully Automated**
Power Management Automation Level: **Off**
Migration Threshold: **Apply priority 1, priority 2, and priority 3 recommendations.**
Power Management Threshold: **N/A**

DRS Recommendations

Apply	Priority	Recommendation	Reason

We now force DRS to run recommendations so that virtual machines are migrated to dc1 hosts.

3.


File Edit View Inventory Administration Plug-ins Help
Home Inventory Datastores and Datastore Clusters
Summary Virtual Machines Hosts Configuration Tasks & Events Alarms Permissions

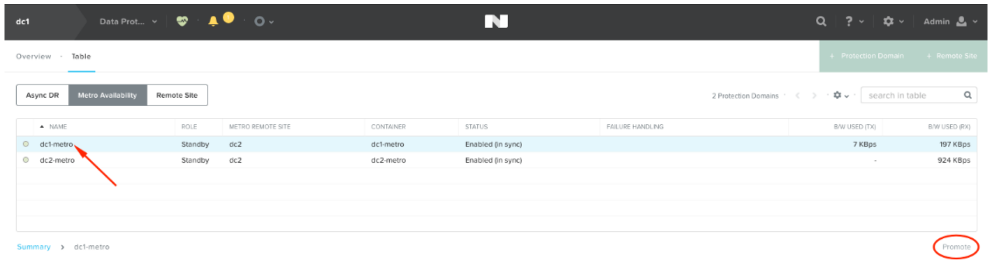
- metro-vcsa.gso.lab
  - metro-dc
    - dc1-local
    - dc1-metro
    - dc2-local
    - dc2-metro
    - NTNX-local-ds-15SM65450286-A
    - NTNX-local-ds-15SM65450286-B
    - NTNX-local-ds-15SM65450286-C
    - NTNX-local-ds-16SM65020018-A
    - NTNX-local-ds-16SM65020018-B
    - NTNX-local-ds-16SM65020018-C

dc1-metro

Name	State	Status	Host
metro-vcsa.gso.lab	Powered On	Normal	dc1nodec.gso.lab
template-dc1	Powered Off	Normal	dc1nodeb.gso.lab
vm1	Powered On	Normal	dc1nodeb.gso.lab
vm10	Powered On	Normal	dc1nodeb.gso.lab
vm2	Powered On	Normal	dc1nodec.gso.lab
vm3	Powered On	Normal	dc1nodeb.gso.lab
vm4	Powered On	Normal	dc1nodec.gso.lab
vm5	Powered On	Normal	dc1nodea.gso.lab
vm6	Powered On	Normal	dc1nodea.gso.lab
vm7	Powered On	Normal	dc1nodea.gso.lab
vm8	Powered On	Normal	dc1nodea.gso.lab
vm9	Powered On	Normal	dc1nodea.gso.lab

We can now double check that all VMs are running on dc1 hosts by selecting the *dc1-metro* datastore in the vSphere client.

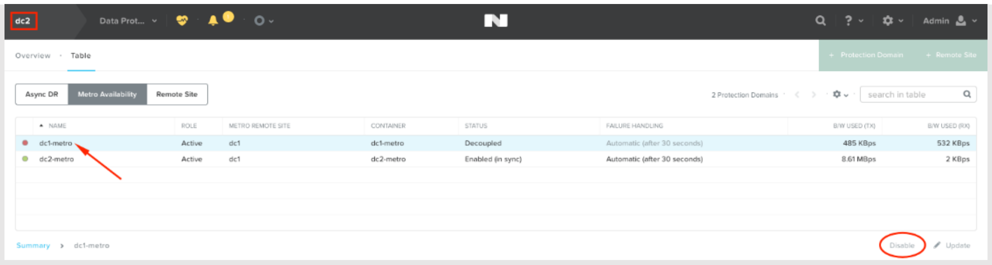
4.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Standby	dc2	dc1-metro	Enabled (in sync)		7 KBps	197 KBps
dc2-metro	Standby	dc2	dc2-metro	Enabled (in sync)		-	924 KBps

We are now ready to promote the *dc1-metro* protection domain on the source site.

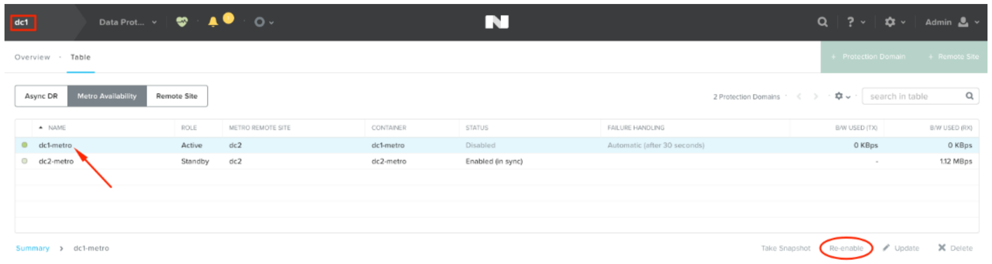
5.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Active	dc1	dc1-metro	Decoupled	Automatic (after 30 seconds)	485 KBps	532 KBps
dc2-metro	Active	dc1	dc2-metro	Enabled (in sync)	Automatic (after 30 seconds)	8.61 MBps	2 KBps

We then disable the *dc1-metro* protection domain on the target site.

6.



	NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
●	dc1-metro	Active	dc2	dc1-metro	Disabled	Automatic (after 30 seconds)	0 KBps	0 KBps
●	dc2-metro	Standby	dc2	dc2-metro	Enabled (in sync)		-	1.12 MBps

Stop

The action described here will overwrite data on dc2 (the target site). You must be absolutely sure that there are no active VMs on dc2 in the *dc1-metro* container before re-enabling replication. If there are active VMs, use storage vMotion to move them to another datastore first.

We can now re-enable replication on the *dc1-metro* protection domain on the source site.

We are done with the failback procedure!

Procedure #2: How to recover a Metro Availability protection domain after a site failure has occurred (unplanned failover and failback)

Overview

You perform an unplanned failover when a disaster has occurred and one of the sites is offline. You want to recover all virtual machines and their storage on the remaining site. For example, assuming you have two datacenters (dc1 and dc2), dc1 has gone offline and therefore you want to recover everything on dc2.

When dc1 comes back online, you will want to move everything back. This is what we call the failback procedure.

The failover process is described using precise terminology. The site which has failed and gone offline is called the primary site (in our example, the primary site is called dc1). The remaining site we are performing recovery on is the recovery site (in our example, the recovery site is called dc2).

For the failback process, we are moving from the recovery site (exp: dc2) back to the primary site (exp: dc1). We do not change site denomination in order to avoid confusion, but you need to be extremely careful about which site is primary and which site is recovery.

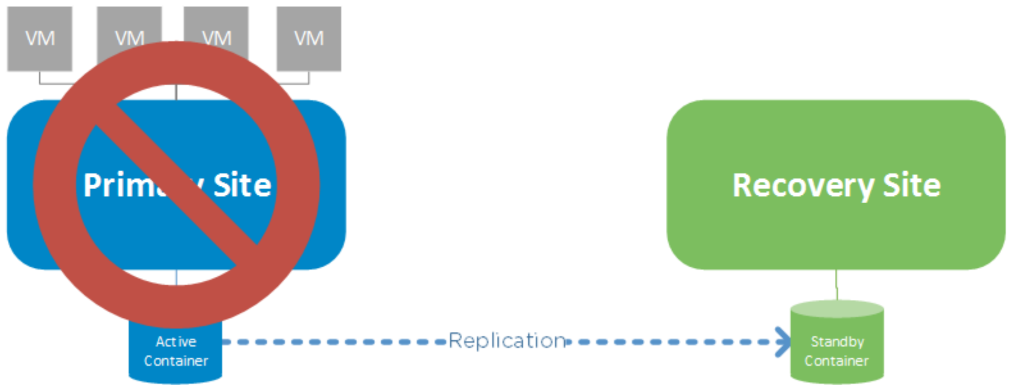
Note that for each process, there is:

1. An initial situation: this shows which primary is source, which is recovery, where VMs are running, where active storage/container is and which direction the replication goes.
2. A process workflow: showing the major steps of the process. Destructive actions are in orange and display a warning logo. When performing those actions, you must make sure your VMs and their data is in the right site, or you may end up overwriting active data with stale data.

A resulting situation: this shows how things look after the procedure has been performed.

Unplanned Failover

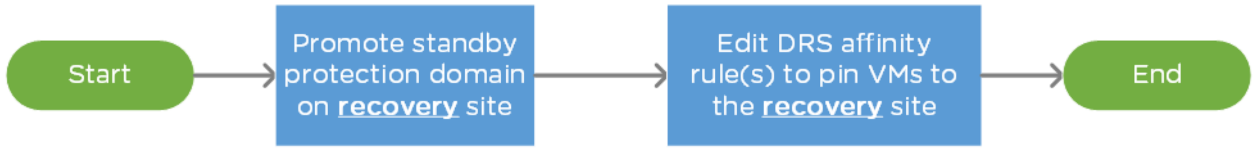
This is the initial situation:



Note that:

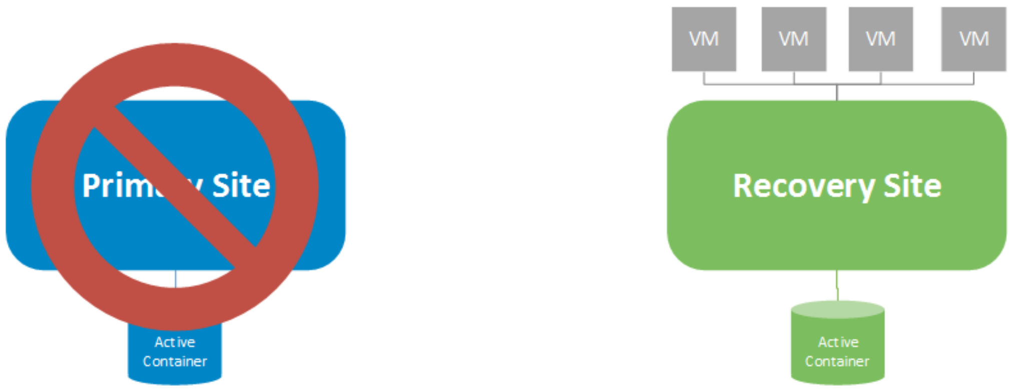
1. The primary site where VMs were running and storage was active has failed and is offline
2. Replication was going from the primary site (dc1) to the recovery site (dc2)

The process workflow is as follows:


Start
Promote standby protection domain on recovery site
Edit DRS affinity rule(s) to pin VMs to the recovery site
End

Detailed steps along with screenshots are included below. This is just an overview workflow to be used as a reference.

The resulting situation will be as follows:



Note that:

1. VMs will run on the recovery site (dc2)
2. Storage will be active on the recovery site (dc2)
3. Replication will not run since dc1 is still offline

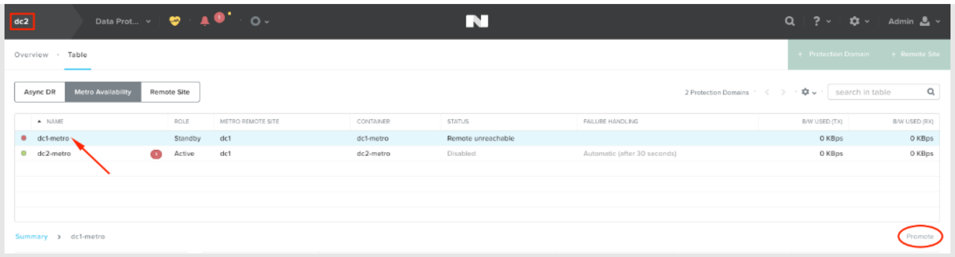
In the detailed steps example below, note that:

1. The **primary** site is called **dc1**.
2. The **recovery** site is called **dc2**.
3. The **container** is called **dc1-metro**.
4. The **protection domain** is also called **dc1-metro**.
5. VMs vm1 to vm10 normally run on dc1.
6. The compute cluster is called metro-cluster.

Stop

Before starting the unplanned failover procedure, you must be absolutely sure that your primary site has failed and gone offline and that VMs are not running on the primary site any longer. Failure to do so can result in data loss.

1.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Standby	dc1	dc1-metro	Remote unreachable		0 KBps	0 KBps
dc2-metro	Active	dc1	dc2-metro	Disabled	Automatic (after 30 seconds)	0 KBps	0 KBps

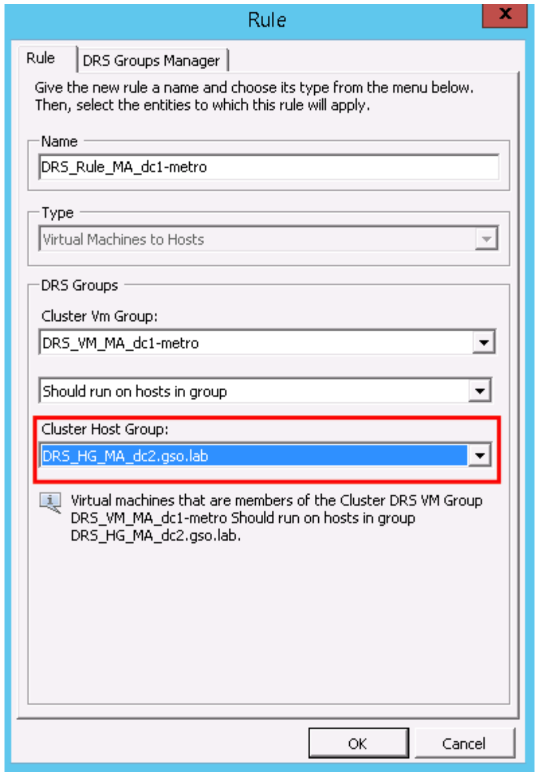
We start by promoting the dc1-metro protection domain on the recovery site from Prism.

At this point, all we have to do for virtual machines to be recovered is wait for VMware HA (high availability) to kick in and restart virtual machines on dc2. This can take several minutes.

Warning

As tempting as it may be, do not try to start manually virtual machines or you will confuse HA.

2.



Give the new rule a name and choose its type from the menu below. Then, select the entities to which this rule will apply.

Name: DRS_Rule_MA_dc1-metro

Type: Virtual Machines to Hosts

DRS Groups

Cluster Vm Group: DRS_VM_MA_dc1-metro

Should run on hosts in group

Cluster Host Group: DRS_HG_MA_dc2.gso.lab

Virtual machines that are members of the Cluster DRS VM Group DRS_VM_MA_dc1-metro Should run on hosts in group DRS_HG_MA_dc2.gso.lab.

We then edit the DRS affinity rules to make sure VMs that are in the dc1-metro container are pinned to hosts in dc2.
Note that you will need to have vCenter running in order to be able to edit the rule. If your vCenter VM was running on dc1, you will have to wait for HA to restart it.

Editing the DRS rule is important because when dc1 comes back online, DRS may try to migrate VMs back to dc1 which will have an active/decoupled version of the storage and this may cause issues where data can be lost and virtual machines will crash.

Note

If this happens, you should immediately shutdown the virtual machines which migrated back to dc1 and migrate them back to dc2 again, then make sure you edit the DRS rule before attempting to power them on again.

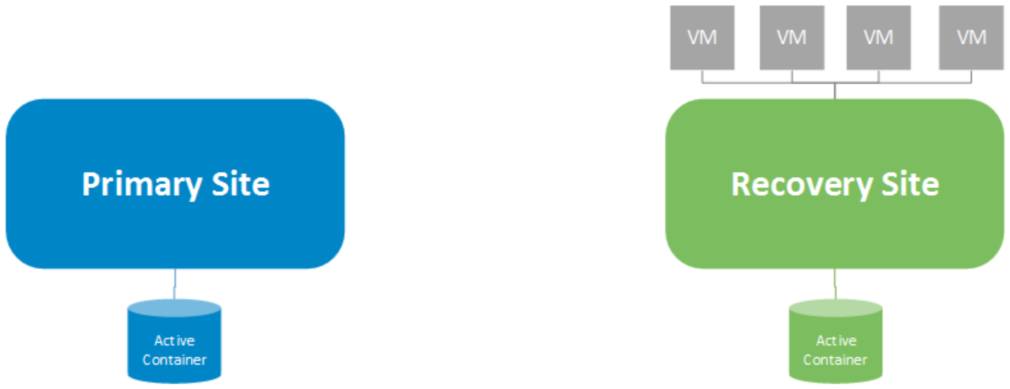
We are done with the unplanned failover procedure!

Unplanned Failback

You should only use this procedure after you had to do an unplanned failover.

Note that using the term “unplanned failback” can seem confusing since you are obviously planning to failback. Unplanned failback only means that you are performing a failback procedure after an unplanned failover.

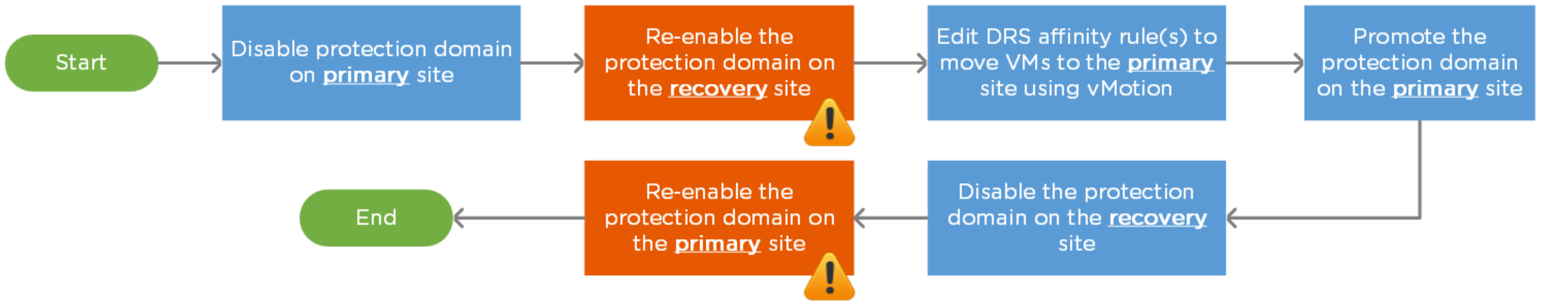
This is the initial situation:



Note that:

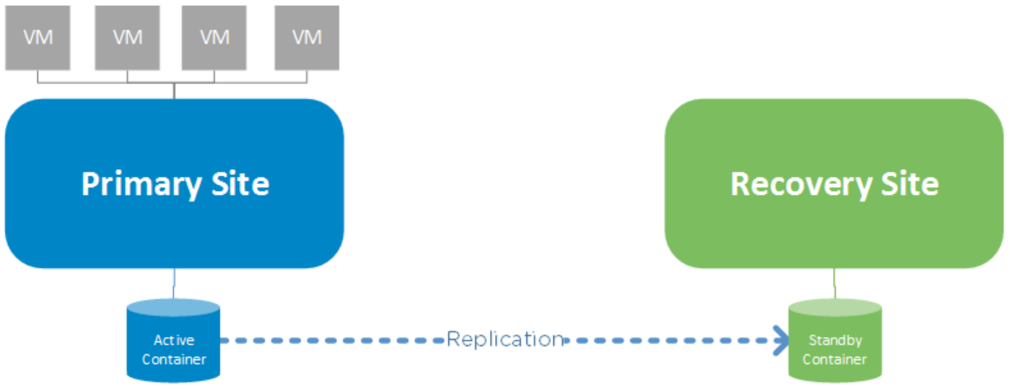
1. VMs are running on the recovery site (dc2) where storage is active.
2. The primary site (dc1) is back online and is running a decoupled active copy of the storage
3. Replication is not enabled

The process workflow is as follows:


Start
Disable protection domain on primary site
Re-enable the protection domain on the recovery site
Edit DRS affinity rule(s) to move VMs to the primary site using vMotion
Promote the protection domain on the primary site
Disable the protection domain on the recovery site
Re-enable the protection domain on the primary site
End

Detailed steps along with screenshots are included below. This is just an overview workflow to be used as a reference.

The resulting situation will be as follows:



Note that:

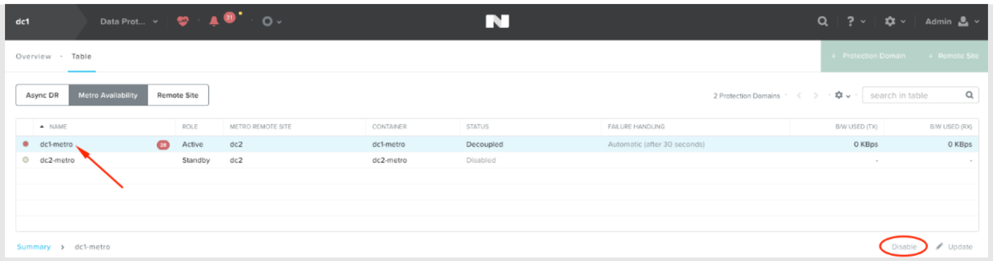
1. VMs will run on the primary site (dc1)

2. Storage will be active on the primary site (dc1) and on standby on the recovery site (dc2)
3. Replication will run from dc1 to dc2

In the detailed steps example below, note that:

1. The **primary** site is called **dc1**.
2. The **recovery** site is called **dc2**.
3. The **container** is called **dc1-metro**.
4. The **protection domain** is also called **dc1-metro**.
5. VMs vm1 to vm10 normally run on dc1.
6. The compute cluster is called metro-cluster.

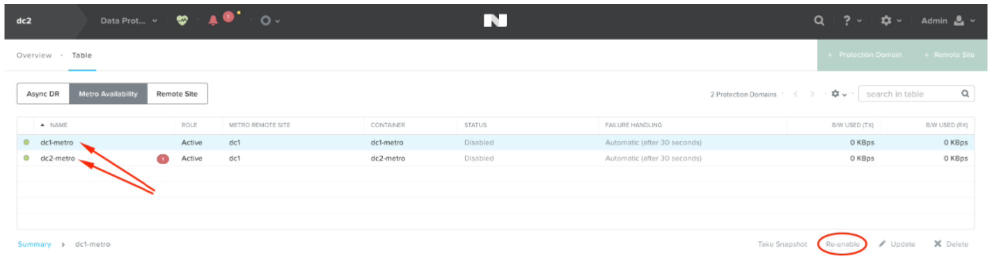
1.



	NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
●	dc1-metro	Active	dc2	dc1-metro	Decoupled	Automatic (after 30 seconds)	0 KBps	0 KBps
○	dc2-metro	Standby	dc2	dc2-metro	Disabled		-	-

The first step is to disable the dc1-metro protection domain on the primary site (dc1) which just came back online.

2.



	NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
●	dc1-metro	Active	dc1	dc1-metro	Disabled	Automatic (after 30 seconds)	0 KBps	0 KBps
●	dc2-metro	Active	dc1	dc2-metro	Disabled	Automatic (after 30 seconds)	0 KBps	0 KBps

Stop

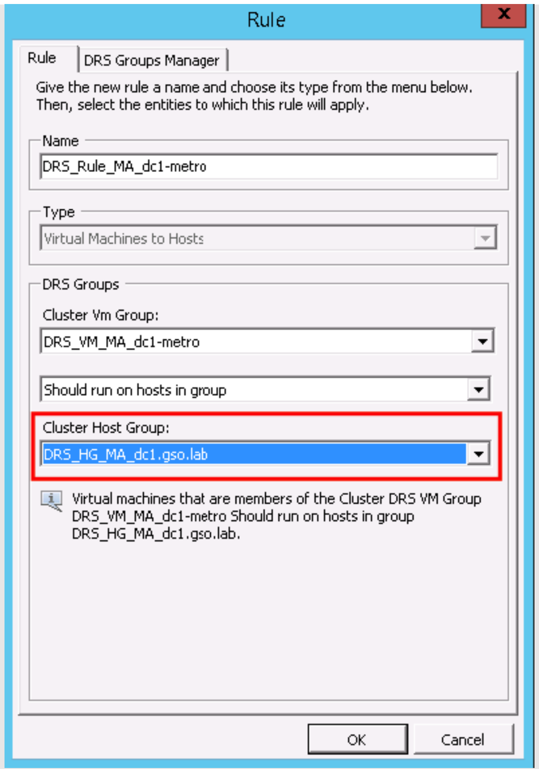
The action described here will overwrite data on dc1, the primary site for the *dc1-metro* container. Before re-enabling replication, make sure there are no virtual machines running in dc1 in the *dc1-metro* container.

We now re-enable replication from the recovery site (dc2) to the primary site (dc1).

If you also had an active protection domain on dc2 which was previously replicating to dc1, you will also need to re-enable replication for that protection domain (in this example, dc2-metro).

Once replication is done, this means that all the data which was modified on dc2 while dc1 was offline is now also replicated to dc1.
We are therefore ready to migrate the VMs back to dc1.

3.



Rule

Rule | DRS Groups Manager

Give the new rule a name and choose its type from the menu below. Then, select the entities to which this rule will apply.

Name: DRS_Rule_MA_dc1-metro

Type: Virtual Machines to Hosts

DRS Groups

Cluster Vm Group: DRS_VM_MA_dc1-metro

Should run on hosts in group

Cluster Host Group: DRS_HG_MA_dc1.gso.lab

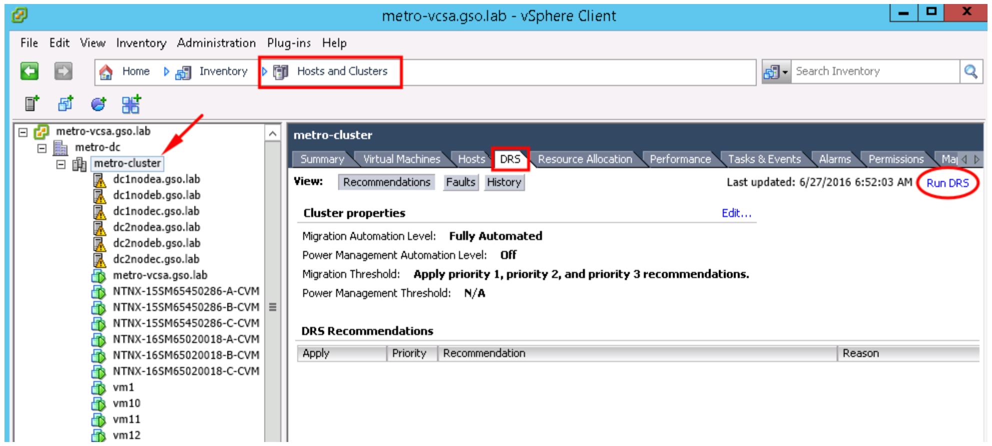
Virtual machines that are members of the Cluster DRS VM Group DRS_VM_MA_dc1-metro Should run on hosts in group DRS_HG_MA_dc1.gso.lab.

Stop

Before doing anything else, wait for the replication to be completed. The *dc1-metro* protection domain replication status should be **Enabled (in sync)**.

We edit the dc1-metro DRS affinity rule so that VMs running in that container are also running on hosts in dc1, the primary site.

4.



Last updated: 6/27/2016 6:52:03 AM

Cluster properties

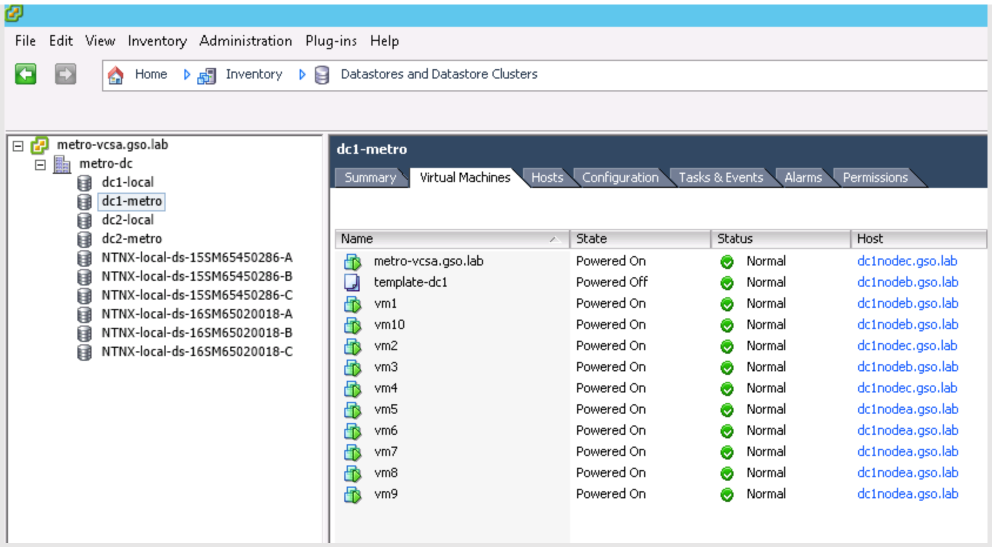
Migration Automation Level: **Fully Automated**
Power Management Automation Level: **Off**
Migration Threshold: **Apply priority 1, priority 2, and priority 3 recommendations.**
Power Management Threshold: **N/A**

DRS Recommendations

Apply	Priority	Recommendation	Reason

We can now force DRS to apply recommendations and migrate the virtual machines to hosts in dc1, the primary site.

5.

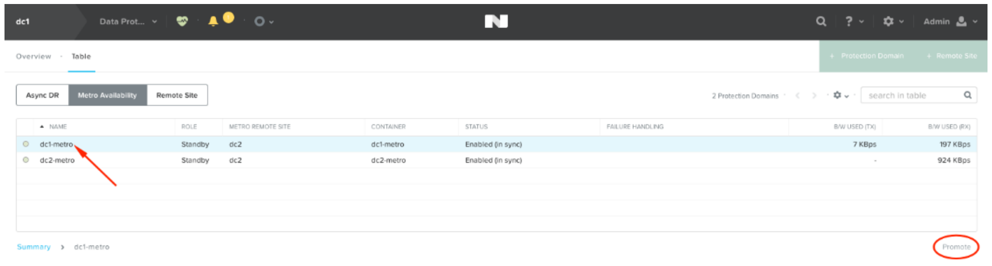

File Edit View Inventory Administration Plug-ins Help
Home Inventory Datastores and Datastore Clusters
metro-vcsa.gso.lab
metro-dc
dc1-local
dc1-metro
dc2-local
dc2-metro
NTNX-local-ds-15SM65450286-A
NTNX-local-ds-15SM65450286-B
NTNX-local-ds-15SM65450286-C
NTNX-local-ds-16SM65020018-A
NTNX-local-ds-16SM65020018-B
NTNX-local-ds-16SM65020018-C
dc1-metro
Summary Virtual Machines Hosts Configuration Tasks & Events Alarms Permissions

Name	State	Status	Host
metro-vcsa.gso.lab	Powered On	Normal	dc1nodec.gso.lab
template-dc1	Powered Off	Normal	dc1nodeb.gso.lab
vm1	Powered On	Normal	dc1nodeb.gso.lab
vm10	Powered On	Normal	dc1nodeb.gso.lab
vm2	Powered On	Normal	dc1nodec.gso.lab
vm3	Powered On	Normal	dc1nodeb.gso.lab
vm4	Powered On	Normal	dc1nodec.gso.lab
vm5	Powered On	Normal	dc1nodea.gso.lab
vm6	Powered On	Normal	dc1nodea.gso.lab
vm7	Powered On	Normal	dc1nodea.gso.lab
vm8	Powered On	Normal	dc1nodea.gso.lab
vm9	Powered On	Normal	dc1nodea.gso.lab

We select the *dc1-metro* datastore and make sure all virtual machines have been successfully migrated to dc1 (the primary site) hosts by DRS.

If any have not been migrated, you will need to migrate them manually using vMotion.

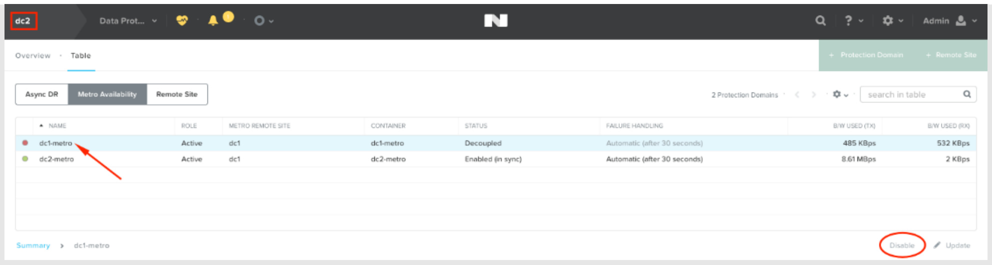
6.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Standby	dc2	dc1-metro	Enabled (in sync)		7 KBps	197 KBps
dc2-metro	Standby	dc2	dc2-metro	Enabled (in sync)		-	924 KBps

We can now promote the *dc1-metro* protection domain on the primary site (dc1).

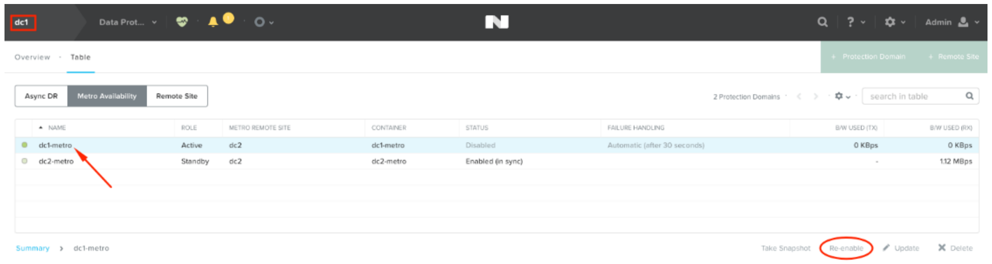
7.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Active	dc1	dc1-metro	Decoupled	Automatic (after 30 seconds)	485 KBps	532 KBps
dc2-metro	Active	dc1	dc2-metro	Enabled (in sync)	Automatic (after 30 seconds)	8.61 MBps	2 KBps

We then disable the dc1-metro protection domain on the recovery site (dc2).

8.



NAME	ROLE	METRO REMOTE SITE	CONTAINER	STATUS	FAILURE HANDLING	B/W USED (TX)	B/W USED (RX)
dc1-metro	Active	dc2	dc1-metro	Disabled	Automatic (after 30 seconds)	0 KBps	0 KBps
dc2-metro	Standby	dc2	dc2-metro	Enabled (in sync)		-	1.12 MBps

Stop

The action described here will overwrite data on dc2, the recovery site for the *dc1-metro* container. Before re-enabling replication, make sure there are no virtual machines running in dc2 in the *dc1-metro* container.

Finally, we can re-enable replication from the primary site (dc1) where virtual machines are running to the secondary site (dc2).

We are done with the unplanned failback procedure!

Appendix

Powershell script to automate DRS affinity rule creation and updates for Metro Availability

The script below can be used to automate initial DRS affinity rules creation for Metro Availability as well as update existing rules (provided they use the script's naming convention).

```
<#
.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 foreach
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
```

Document Revision History

Version	Date	Editor	Description
1.0	July 11th 2016	Stéphane Bourdeaud (stephane.bourdeaud@nutanix.com)	Initial creation of the document

The latest version of this document can be found in the Nutanix Services Consulting repository.