Virtual Domain Controller Cloning in Windows Server 2012

Tom Moser [MSFT]
1 Oct 2012 5:00 AM

Tom Moser here with a post on one of the new ADDS features in Windows Server 2012; Virtual Domain Controller Cloning.

Until now, cloning, snapshotting, copying, or pretty much doing anything but rebuilding from scratch to a virtual domain controller wasn't just unsupported; it had the potential to be really bad for your directory. Cloning or restoring snapshots of DCs could result in USN rollbacks or lingering objects, just to name a couple of problems.

Starting in Windows Server 2012, we now support DC cloning as well as snapshot restoration of domain controllers. With the RTM bits available, I found myself rebuilding my lab and took the opportunity to document the process to demonstrate just how easy it is to clone virtual domain controllers with Windows Server 2012.

Requirements

There are a few base infrastructure requirements to take advantage of DC cloning.

- The hypervisor must support VM-GenerationID. Hyper-V running on Windows Server 2012 supports this feature. Other virtualization vendors will have the ability to implement this as well, so check with your vendor to see if it's supported.
- The source virtual DC must be running Windows Server 2012.
- The PDC emulator role holder must be online and available to the cloned DC *and* must be running Windows Server 2012.

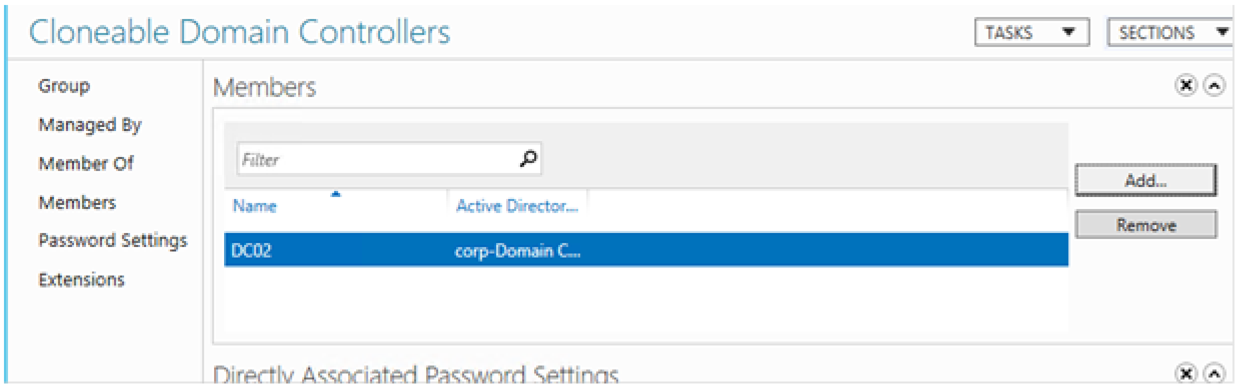
There are a few other steps and requirements and I'll take you through those now.

Cloneable Domain Controllers Group

There's a new group in town. It's called **Cloneable Domain Controllers** and you can find it in the Users container. Membership in this group dictates whether a DC can or cannot be **cloned.** This group has some permissions set on the domain head that **should not** be removed. Removing these permissions will cause cloning to fail. Also, as a best practice, DCs shouldn't be added to the group until you plan to clone and DCs should be removed from the group once cloning is complete. **Cloned DCs will also end up in the Cloneable Domain Controllers group.** Make sure to remove those as well.

In the case of my lab, I planned to build out a whole bunch of Server Core DCs. I built a single DC running the GUI-enabled version of Server, then built a DC running Server Core. This DC, DC02, will be my source domain controller.

The first step in the cloning process is to add the source DC to the Cloneable Domain Controllers group. Here, I've used the latest version of Active Directory Administrative Center to add my DC to the group. Make sure to select "Computers" under Object Type in the object picker when adding the DC. Here, I verified that the group now contains my source DC, DC02. (Figure 1).



Name	Active Director...
DC02	corp-Domain C...

Figure 1 - Cloneable Domain Controllers

We're almost there. No kidding. Next, we need to create the config file.

DCCloneConfig.xml

There's one key difference between a cloned DC and a DC that is being restored to a previous snapshot: DCCloneConfig.XML.

DCCloneConfig.xml is an XML configuration file that contains all of the settings the cloned DC will take when it boots. This includes network settings, DNS, WINS, AD site name, new DC name and more. This file can be generated in a few different ways.

- The *New-ADDCCloneConfig* cmdlet in PowerShell
- By hand with an XML editor
- By editing an existing config file, again with an XML editor (Notepad is not an XML editor.)

In my lab, I used the PowerShell cmdlet to generate the config. This is an easy, safe way to generate the file and ensures there won't be any issues during cloning.

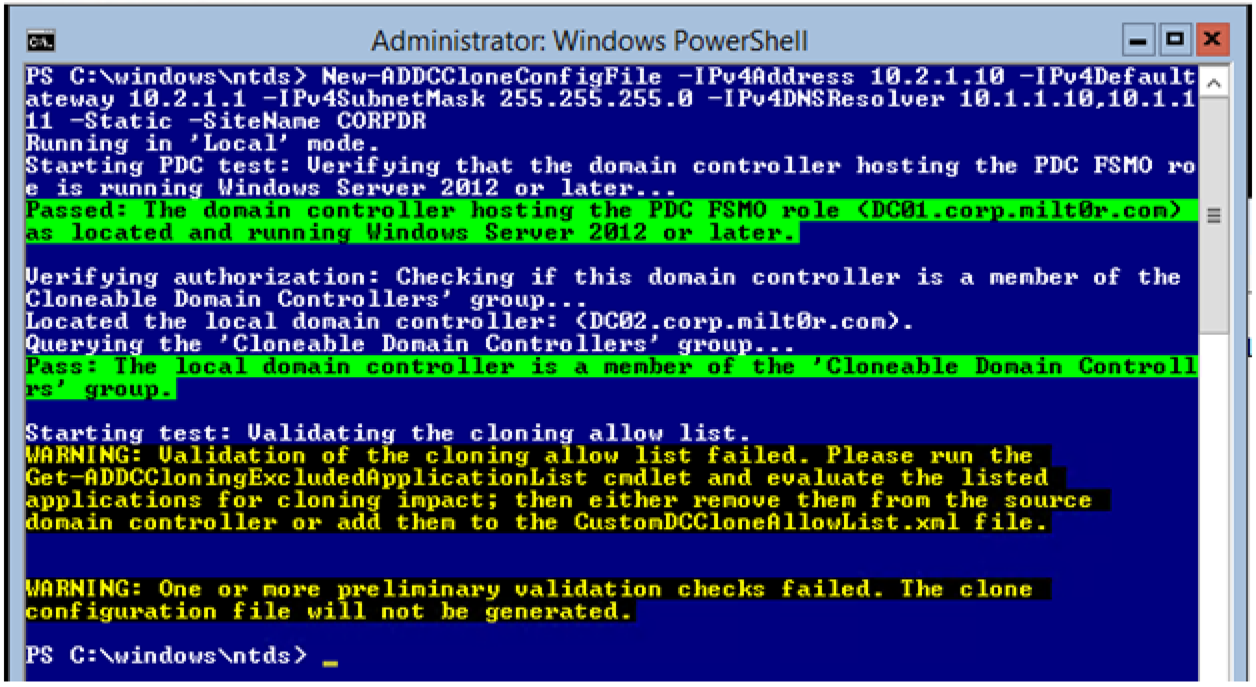
On your Windows Server 2012 domain controller, fire up PowerShell. From there, take a minute to run *Get-Help New-ADDCCloneConfig –Full* and review the help page.

Now that you've thoroughly reviewed the help page, let's go through the steps. On DC02, I ran the following:

```
New-ADDCCloneConfigFile -IPv4Address 10.2.1.10 -IPv4DefaultGateway 10.2.1.1 -
IPv4SubnetMask 255.255.255.0 -IPv4DNSResolver 10.1.1.10,10.1.1.11 -Static -SiteName
CORPDR
```

This cmdlet will generate a config XML using the specified parameters, in this case IP information, AD site to use, and that it's a static IP configuration. The IP information provided to the cmdlet will be used by the new cloned domain controller when it begins cloning.

The cmdlet is going to do a few things prior to generating the configuration XML file. First, it's going to verify that the PDC is available and running Windows Server 2012 or later (one of the pre-reqs above). This check can be skipped by specifying the –*Offline* switch. You should only need to do this if a global catalog isn't available. Second, it's going to verify that the source DC (where you're running the cmdlet) is a member of the *Cloneable Domain Controllers* group. Finally, the cmdlet is going to check the DC for applications that may not support cloning. If any applications or services are detected (such as DHCP), generation of the config will **fail.** You'll see something like Figure 2 (below).

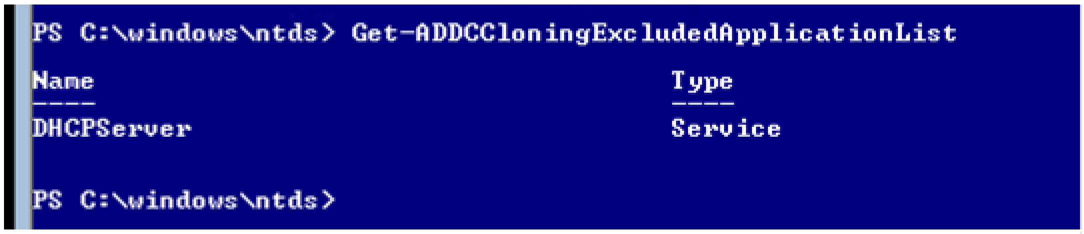


```
PS C:\windows\ntds> New-ADDCCloneConfigFile -IPv4Address 10.2.1.10 -IPv4Default
ateway 10.2.1.1 -IPv4SubnetMask 255.255.255.0 -IPv4DNSResolver 10.1.1.10,10.1.1
11 -Static -SiteName CORPDR
Running in 'Local' mode.
Starting PDC test: Verifying that the domain controller hosting the PDC FSMO ro
e is running Windows Server 2012 or later...
Passed: The domain controller hosting the PDC FSMO role (DC01.corp.milt0r.com)
as located and running Windows Server 2012 or later.

Verifying authorization: Checking if this domain controller is a member of the
Cloneable Domain Controllers' group...
Located the local domain controller: (DC02.corp.milt0r.com).
Querying the 'Cloneable Domain Controllers' group...
Pass: The local domain controller is a member of the 'Cloneable Domain Controll
rs' group.

Starting test: Validating the cloning allow list.
WARNING: Validation of the cloning allow list failed. Please run the
Get-ADDCCloningExcludedApplicationList cmdlet and evaluate the listed
applications for cloning impact; then either remove them from the source
domain controller or add them to the CustomDCCloneAllowList.xml file.


WARNING: One or more preliminary validation checks failed. The clone
configuration file will not be generated.

PS C:\windows\ntds> _
```

Figure 2 - Failed Clone Config Generation

The cmdlet warning text is pretty helpful. It tells us that applications were found that are not on the allowed list. What's on the allowed list? Good question. You can find it at *C:\Windows\System32\DefaultDCCloneAllowList.xml*. This list exists by default on Windows Server 2012 domain controllers and should **not** be modified. If you want to add a discovered application to the list, you'll need to generate a custom DC allow list.

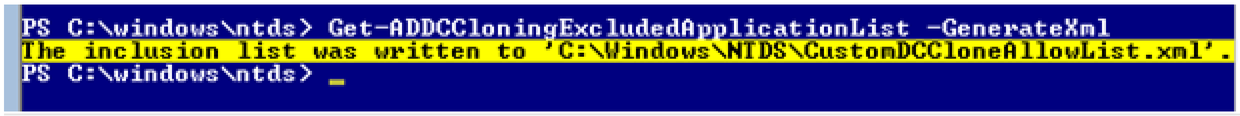
Prior to generating the custom allow list, you'll want to review what applications actually caused the warning. Running *Get-ADDCCloningExcludedApplicationList* (tab complete is your buddy here) will show you that list. This cmdlet will dump out all of the services that were discovered while trying to generate the DCCloneConfig.XML (Figure 3). From here, you can make a decision; remove the discovered application or service *or* add the application to the custom allow list. Generally speaking, if it's a Microsoft service that has been flagged, as DHCP has been flagged in this example, you should not clone it. If a third party application is discovered and you've verified with the vendor that cloning is supported, then you can jump to Figure 4.



```
PS C:\windows\ntds> Get-ADDCCloningExcludedApplicationList

Name                                             Type
----                                             ----
DHCPServer                                       Service


PS C:\windows\ntds>
```

Figure 3 - We know who to blame...

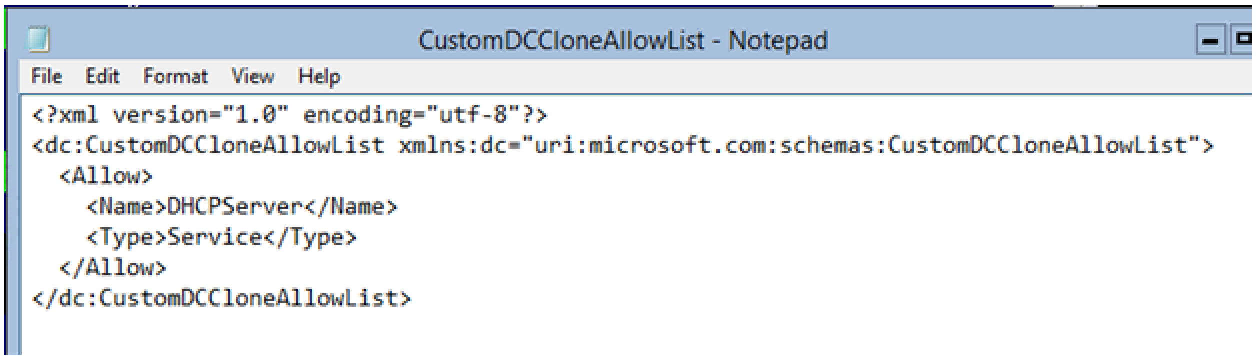
You've verified with the application vendor that the service is OK to clone. Generating the allow list is easy. Run *Get-ADDCCloningExcludedApplicationList* again, but specify the –*GenerateXML* switch.



```
PS C:\windows\ntds> Get-ADDCCloningExcludedApplicationList -GenerateXml
The inclusion list was written to 'C:\Windows\NTDS\CustomDCCloneAllowList.xml'.
PS C:\windows\ntds> _
```

Figure 4 - That was easy...

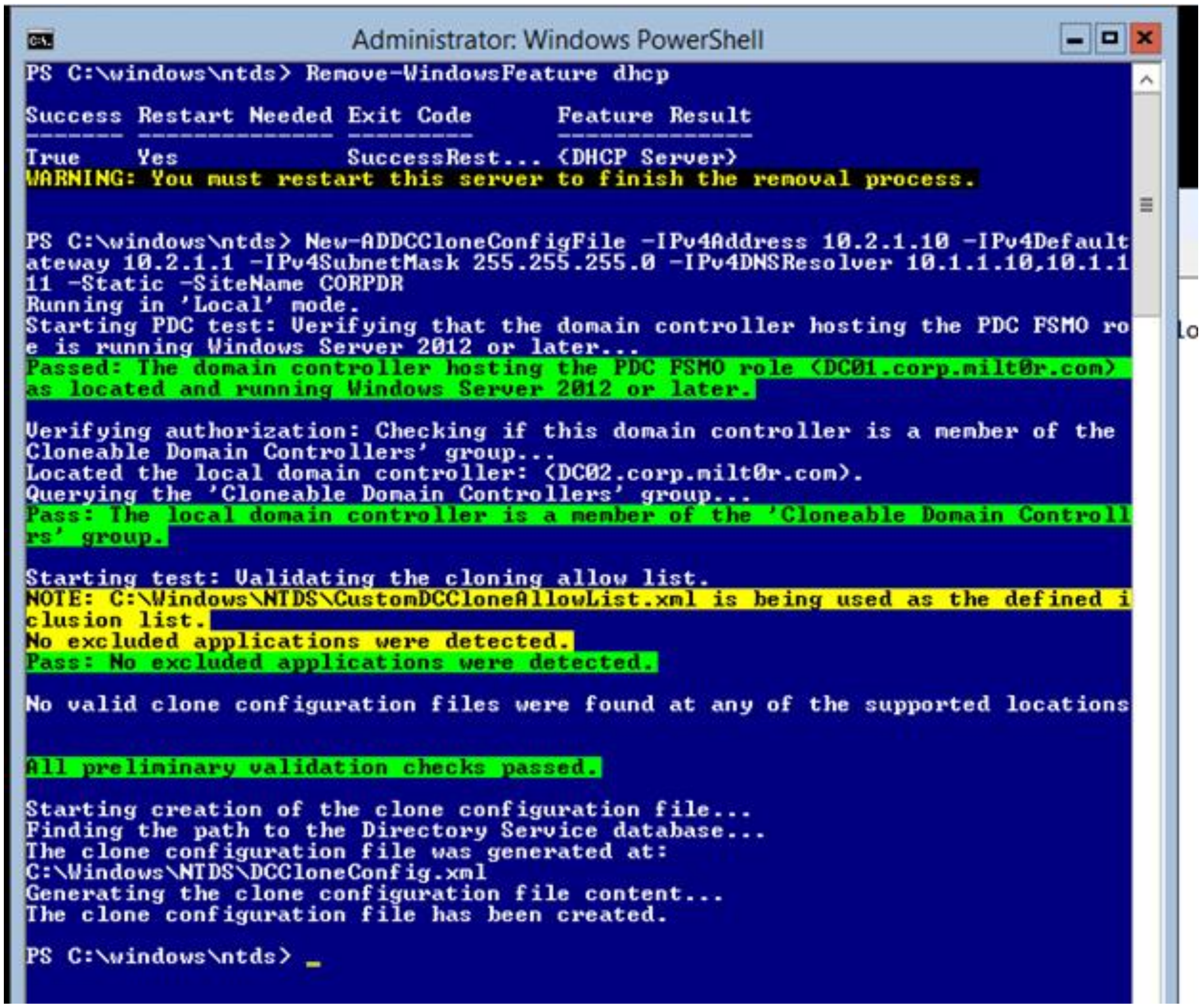
The cmdlet generates the XML and writes it out to c:\windows\ntds, as the output shows. Viewing the contents in notepad shows this (Figure 5):



```
<?xml version="1.0" encoding="utf-8"?>
<dc:CustomDCCloneAllowList xmlns:dc="uri:microsoft.com:schemas:CustomDCCloneAllowList">
  <Allow>
    <Name>DHCPServer</Name>
    <Type>Service</Type>
  </Allow>
</dc:CustomDCCloneAllowList>
```

Figure 5 - Or you should just remove DHCP...

And that's what the custom allow list looks like. Again, you could use an XML editor to modify this, but should probably remove the application unless you have a specific reason to create a clone of the service **and** you've verified that the application supports cloning. Once you've generated the XML file, you can run *New-ADDCCloneConfigFile* again and it'll successfully generate your config. In my case, I used *Remove-WindowsFeature* and removed DHCP from my DC. Figure 6 shows the PowerShell commands for both removing DHCP as well as successfully generating my DCCloneConfig.



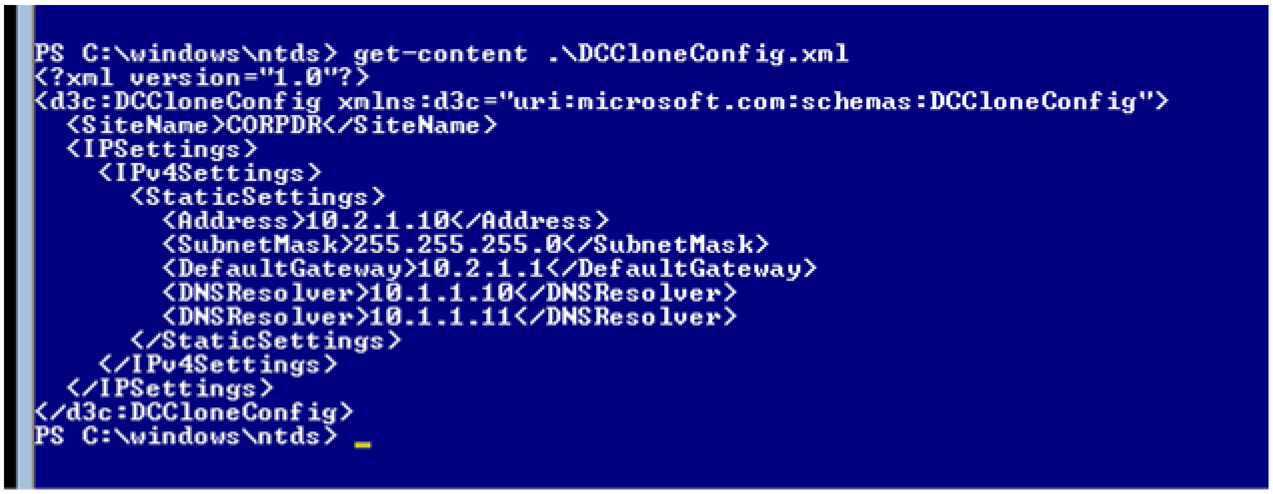
```
Administrator: Windows PowerShell
PS C:\windows\ntds> Remove-WindowsFeature dhcp

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    Yes            SuccessRest... {DHCP Server}
WARNING: You must restart this server to finish the removal process.


PS C:\windows\ntds> New-ADDCCloneConfigFile -IPv4Address 10.2.1.10 -IPv4Default
ateway 10.2.1.1 -IPv4SubnetMask 255.255.255.0 -IPv4DNSResolver 10.1.1.10,10.1.1
11 -Static -SiteName CORPDR
Running in 'Local' mode.
Starting PDC test: Verifying that the domain controller hosting the PDC FSMO ro
e is running Windows Server 2012 or later...
Passed: The domain controller hosting the PDC FSMO role (DC01.corp.milt0r.com)
as located and running Windows Server 2012 or later.

Verifying authorization: Checking if this domain controller is a member of the
Cloneable Domain Controllers' group...
Located the local domain controller: (DC02.corp.milt0r.com).
Querying the 'Cloneable Domain Controllers' group...
Pass: The local domain controller is a member of the 'Cloneable Domain Controll
rs' group.

Starting test: Validating the cloning allow list.
NOTE: C:\Windows\NTDS\CustomDCCloneAllowList.xml is being used as the defined i
clusion list.
No excluded applications were detected.
Pass: No excluded applications were detected.

No valid clone configuration files were found at any of the supported locations


All preliminary validation checks passed.

Starting creation of the clone configuration file...
Finding the path to the Directory Service database...
The clone configuration file was generated at:
C:\Windows\NTDS\DCCloneConfig.xml
Generating the clone configuration file content...
The clone configuration file has been created.

PS C:\windows\ntds> _
```

Figure 6 - That's a lot of green, so I think it's OK.

As the output shows, the XML file is written to c:\windows\ntds. That's one of **three** valid locations where the file can be placed for cloning. All three locations are:

- %windir%\NTDS
- Wherever the DIT lives (if you've changed the path to D:\NTDS, for example)
- The root of any removable media

The config file should have been written out to c:\windows\ntds. Viewing the content with *Get-Content* in PowerShell shows us what the config looks like (Figure 7). If you're familiar with XML-based configuration files, this should look pretty typical.



```
PS C:\windows\ntds> get-content .\DCCloneConfig.xml
<?xml version="1.0"?>
<d3c:DCCloneConfig xmlns:d3c="uri:microsoft.com:schemas:DCCloneConfig">
  <SiteName>CORPDR</SiteName>
  <IPSettings>
    <IPv4Settings>
      <StaticSettings>
        <Address>10.2.1.10</Address>
        <SubnetMask>255.255.255.0</SubnetMask>
        <DefaultGateway>10.2.1.1</DefaultGateway>
        <DNSResolver>10.1.1.10</DNSResolver>
        <DNSResolver>10.1.1.11</DNSResolver>
      </StaticSettings>
    </IPv4Settings>
  </IPSettings>
</d3c:DCCloneConfig>
PS C:\windows\ntds> _
```

Figure 7 - XML editor. Not Notepad.

Now I've got the config file, I've removed the unsupported service and I'm ready to start the cloning process.

Copying the Source DC

The last step now is to export the source virtual machine. This can be accomplished via PowerShell or the Hyper-V management console.

First, turn off the source DC then export the VM. I used PowerShell (Figure 8). **Milt0rDC02** is the name I used in Hyper-V for **DC02**.

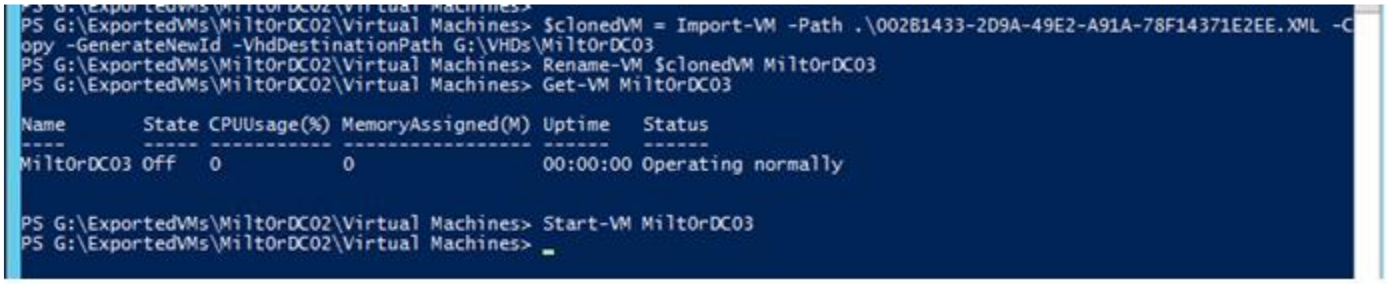
Figure 8 - This is going to take a minute

Now that I've exported the source DC, I need to import the VM. In PowerShell, I navigate to the path were I exported the VM and to the Virtual Machines subdirectory. There, I run Import-VM and store the result in a variable called *$clonedVM*. **Milt0rDC03** is the name in Hyper-V I'll use for this new domain controller.

```
  $clonedVM = Import-VM -Path <VM XML Path>.xml -Copy -GenerateNewId -
VhdDestinationPath G:\VHDs\Milt0rDC03
```

With the Import-VM cmdlet (Figure 9), I specify the path to the VM XML file and use the –*Copy* and –*GenerateNewId* switches. The copy switch is going to leave my original export intact and the GenerateNewId switch will generate a VM ID that differs from the ID of the originally exported machine. I also had to specify a different VHD path using –*VhdDestinationPath*. This path is the directory where the VHD will be copied to. **The VHD will import using the same name as the VHD from the export.** In my case, this was *Milt0rDC02.vhdx*. The copied VHD will import to *G:\VHDs\Milt0rDC03\Milt0rDC02.vhdx*. These can be renamed and the VM reconfigured after the import is complete. You could also simply create a copy of the VHD, rename it, create a new VM, then attach the VHD to the new VM.

Once that completes, I'm able to quickly rename the VM (as it will retain the name of the originally exported VM) by using the *Rename-VM* cmdlet (see Figure 9).



```
PS G:\ExportedVMs\Milt0rDC02\Virtual Machines> $clonedVM = Import-VM -Path .\002B1433-2D9A-49E2-A91A-78F14371E2EE.XML -C
opy -GenerateNewId -VhdDestinationPath G:\VHDs\Milt0rDC03
PS G:\ExportedVMs\Milt0rDC02\Virtual Machines> Rename-VM $clonedVM Milt0rDC03
PS G:\ExportedVMs\Milt0rDC02\Virtual Machines> Get-VM Milt0rDC03

Name       State CPUUsage(%) MemoryAssigned(M) Uptime   Status
----       ----- ----------- ----------------- ------   ------
Milt0rDC03 Off   0           0                 00:00:00 Operating normally


PS G:\ExportedVMs\Milt0rDC02\Virtual Machines> Start-VM Milt0rDC03
PS G:\ExportedVMs\Milt0rDC02\Virtual Machines> _
```

Figure 9 - Aaand done.

That's it. From here, it's just a matter of turning on the DC and letting everything happen on its own.

Cloning

If you've made it this far in the post, you're probably thinking "this seems pretty long and drawn out." Sure, I'm pretty long-winded, but we've actually only done two things here: Create a clone config file (which is fast, assuming you don't have any apps that aren't allowed) and create a copy of the source DC. Now we can power on the copied DC, our clone, and the rest should take care of itself.

I fire up the VM and after a minute or so of "Completing installation" I see this screen (Figure 10).

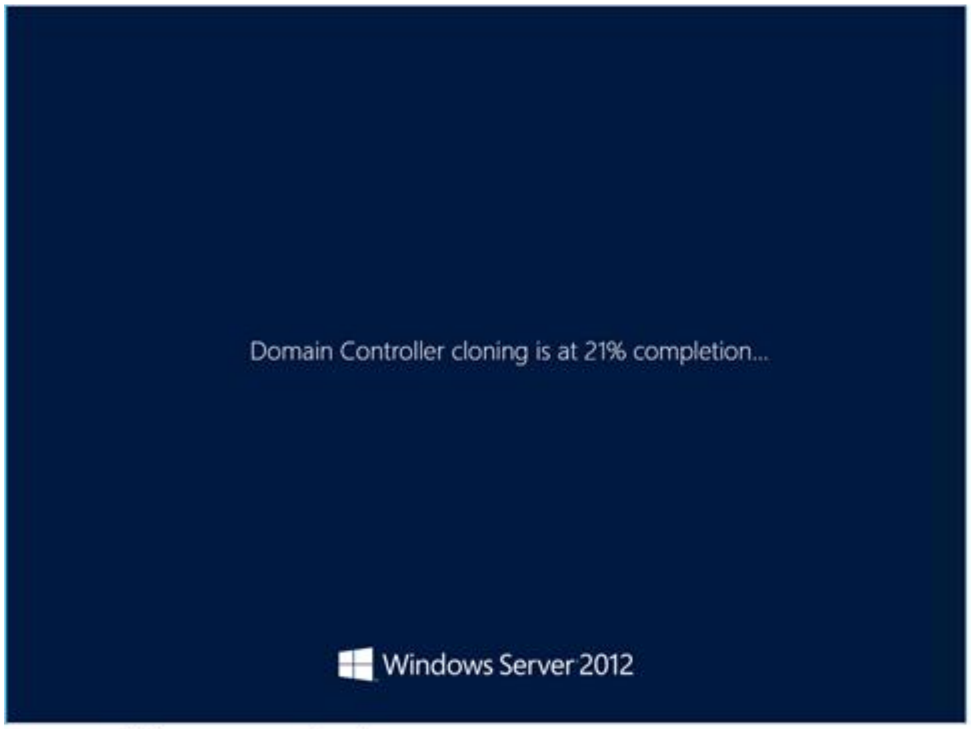

Domain Controller cloning is at 21% completion...

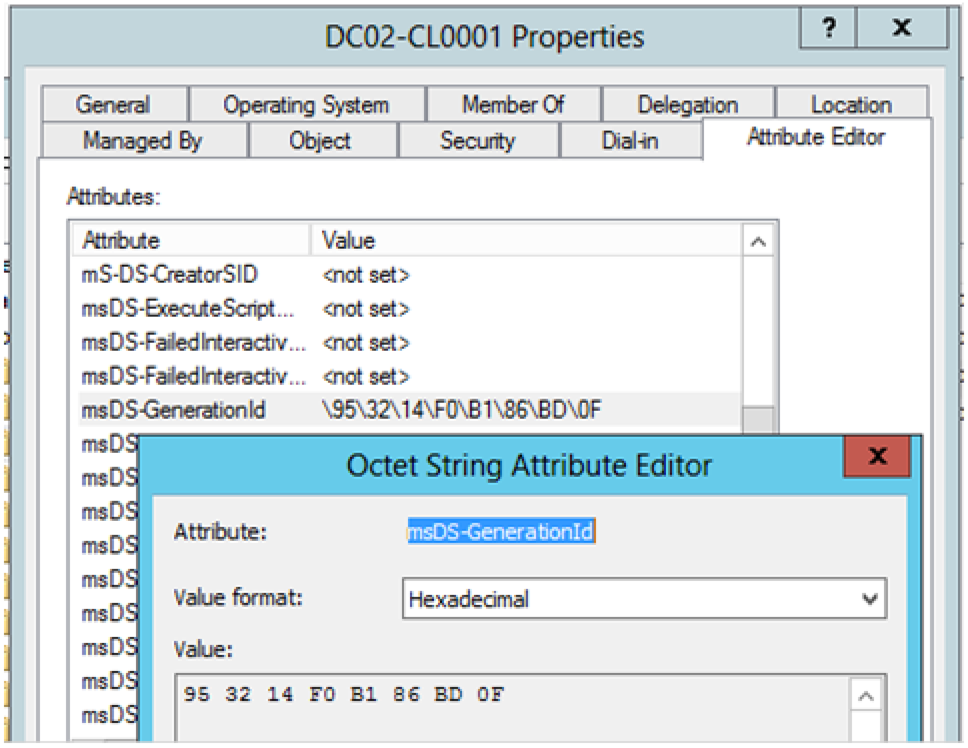
Figure 10 - I'll clone an ARMY of DCs!

A few minutes of this and the DC will reboot. When it's up, you should see a new DC in the Domain Controllers OU and logging in to the DC should indicate that you're in normal mode and not DSRM. Now you'll want to take some time to verify that both NTDS and SYSVOL replication are working properly and that everything is responding as expected. If it checks out, you've successfully cloned your virtual domain controller.

VM-GenerationID

At the beginning of the post, I mentioned that the hypervisor must support **VM-GenerationID**. VM-GenerationID is a property that is exposed to the VM via the virtualization drivers and is unique to that virtual machine. Each Windows Server 2012 domain controller stores its own VM-GenerationID in a property called *msDS-GenerationId* on the DC's computer object. **This value does not replicate.** If you try to view the value of msDS-GenerationId from another DC, you'll see that the value shows up as not set.

Figure 11, below, shows the value on DC02-CL0001. I had to connect directly to that DC in ADUC to see the value.



Attribute	Value
mS-DS-CreatorSID	<not set>
msDS-ExecuteScript...	<not set>
msDS-FailedInteractiv...	<not set>
msDS-FailedInteractiv...	<not set>
msDS-GenerationId	\95\32\14\F0\B1\86\BD\0F

Attribute: msDS-GenerationId

Value format: Hexadecimal

Value:

```
95 32 14 F0 B1 86 BD 0F
```

Figure 11 - msDS-GenerationId as seen on DC02-CL0001

Each time a Windows Server 2012 domain controller boots, it compares the VM-GenerationID presented by the hypervisor to the value it has stored in the DIT. If they match, it boots normally. If they don't match, this indicates that the DC is a restored snapshot or a clone.

How does it determine if the VM is a restored snapshot or a clone?

Remember that DCCloneConfig.xml we created earlier? When the VM-GenerationID mismatch is detected, the next step is to check for that configuration file. If it exists, the cloning process starts. If that file doesn't exist, it's assumed that this is a restored snapshot and the restore process starts.

After, there's a bunch of "stuff" that happens which, for the sake of brevity, I'll cover in another post. The DC reads the config, modifies the domain controller configuration based on the settings in the DCCloneConfig.xml, and attempts to complete the cloning process. Here's where you might run in to issues, especially if you cheated and just dropped a config on another server without running the New-ADDCCloneConfig cmdlet. Some problems you might run in to are:

- PDC is unavailable or not running Server 2012. I actually ran in to this because I neglected to change my virtual switch in Hyper-V.
- You thought you were being sneaky and dropped a config file on to a clone source VHD with apps that are not in the default allowed list, without adding a custom allow list. This is checked during cloning and will cause the clone to fail.
- A duplicate IP is detected.

In each of those cases, the clone will fail and the DC will boot to Directory Services Restore Mode. Logging on to the DC and viewing the DCPromo.log file in C:\Windows\Debug should give you some hints on why it failed.

Assuming you didn't run in to any of those issues and your DC clone completed successfully, you're all done! If you didn't specify the *–CloneComputerName* parameter when you ran New-ADDCCloneConfig (which I intentionally left out), the DC will boot using a generated name. If you do specify that parameter, the given name will be used for the new domain controller. In the screenshot above (Figure 11), you can see that my DC is called DC02-CL0001. The source DC was called DC02. The cloning process will take the first **eight** characters of the source domain controller's name then append **–CL000X.** This value will count up for each clone. Once you get to 9999, you'll need to pick a new name to clone immediately followed by calling your TAM to brag about having 10,000 domain controllers. Since I don't plan on continuing to clone, I decided to rename the domain controller to **DC03** and called it a day.

Wrap-up

If you've made it this far, you'll now be able to successfully clone DCs in your environment. After each time through cloning, I mounted the exported VHD from the source VM, modified the DCCloneConfig.xml file, and created another copy of the VM. This process enabled me to very quickly create four copies of my original Server Core DC. I'll demo that in a future post.

I hope this post gives you an understanding of virtual domain controller cloning and the steps involved to make it work. In a near-future post, I'll cover the process with a little more depth as well as troubleshooting and safe restore. Thanks for reading!

-Tom Moser