


[Home](#) [Archives](#) [Profile](#) [Subscribe](#)
[Main](#)

02/20/2012

A thought experiment - The Universal Fabric

This post is in response to a [post by Scott Lowe](#), another [post by Maish Saidel-Keesing](#) and his reaction to [TDZ](#) as a potential solution to satisfy his requirements for a “dvFabric”. Maish’s statement via twitter:

“...but this (TDZ) only covers block storage. What about NAS?”

honestly set me back on my heels a bit. My first reaction was “TDZ and NAS are orthogonal and will operate independently from one another”, but as I thought about his point some more, I realized that while FC/FCoE and NAS are completely different protocols, why can’t a “TDZ like” approach apply to both? And this led to another question, why should the solution be limited to VMware users? Then I figured since we’re thinking “outside of the box” why not think **way outside** of the box...

Before I get too much further along, I feel it’s important to acknowledge the contributions of Mark Lippitt, Jack Harwood, David Black, David Cohen, Wayne Berthiaume and Mugdha Kulkarni especially in the iSCSI portion of this post. I have a hard time figuring out exactly where their ideas end and mine begin but suffice it to say much of what I’m describing came about due to some amount of collaboration with this team.

I need more storage!

When I setup a new configuration, the [steps that I have to perform](#) to make additional storage visible to the host vary based on the OS and protocol in use. Perhaps it’s a bit obvious, but successfully performing these steps is not the goal, they are the means to an end. I’ll go out on a limb and say if there’s an easier way to accomplish the same set of tasks, it would be very interesting to almost everyone. Breaking this idea down to its most basic component, you start with an administrator who’s thinking “Oh, <optional expletive> I need more storage!” *Actually, ideally, applications would automatically provision additional storage capacity as needed and notify the administrator that it was done, but I digress.* My point is, the set of steps that the Administrators have to perform to provision the additional storage are, in reality, nothing more than obstacles getting in the way of the real goal of **provisioning more storage**.


[Erik Smith](#)

ARCHIVES

[September 2022](#)
[August 2020](#)
[April 2019](#)
[December 2017](#)
[November 2017](#)
[July 2017](#)
[April 2017](#)
[September 2016](#)
[July 2016](#)
[February 2016](#)

SEARCH

CATEGORIES

[Subscribe to this blog's feed](#)

DISCLAIMER

This is not an official EMC blog. The opinions expressed here are my personal opinions. Content published here is not read or approved in advance by EMC and does not necessarily reflect the views and opinions of EMC nor does it constitute any official communication of EMC.

Therefore, reducing the number of storage provisioning steps to be as close to zero as possible would be viewed as a good thing by end users. If we can agree on this, then there are at least two general approaches to satisfying this requirement; I'll refer to them as "top down" and "bottom up" (I know, not very original).

Top down

Top down is the idea of using a centralized "Controller" to configure the Compute, Network and Storage pools. For more information on this kind of approach, check out [OpenStack](#) and the numerous IaaS, PaaS offerings available today that meet this kind of requirement. I'm a huge fan of OpenStack and I'm devoting a significant portion of my personal time getting to know it better (IOW, expect a post or two about it in the future). However, one of my concerns about the OpenStack approach is it's so radically different from what has been done traditionally. I mean if you're running a production data center today and are looking for a way to **incrementally** improve the efficiency of the storage provisioning process, you might look at something like OpenStack as a potential solution that has no business being anywhere near your production data center (for now). That having been said, one example of how a top down approach could be used would be:

1. The user interacts with the centralized controller to provision additional storage.
2. The controller would then parse the users request, potentially apply some rules to the request and then interact with the host, network and storage to:
 - a. Provision the storage and allow the host access to a certain number of LUNs
 - b. Configure the network to allow the host and storage to access each other
 - c. Point the host at the proper storage port

Long term I think this approach has a ton of potential to make our lives easier.

Bottom up

Bottom up is the idea of configuring one of the end devices and then using a centralized service to reduce the number of manual configuration steps required on the host, network and storage. One example of this approach is [TDZ \(Target Driven Zoning\)](#). Although TDZ was designed for to be used in an FC/FCoE environment, I'll describe over the next few sections how the same concepts could be applied to iSCSI and NAS.

FC/FCoE

With the current version of TDZ, the basic idea is to extend existing (and proven) features and protocols to **incrementally improve** the FC/FCoE end users experience **while minimizing risk**. It does this by allowing a storage administrator to specify which hosts have access to a certain set of LUNs (as is done today) with the incremental improvement being:

1. the Storage port is allowed to "publish" a list of "allowed" hosts to a centralized point (the fabric zone server)
2. the FC/FCoE network elements use this list to automatically configure itself (update zoning)

3. the hosts automatically discover the correct storage (via RSCN)

To make a long story short, the only change required to FC/FCoE to make this happen was the creation of the “peer zone”. This very small change enabled a significant saving in terms of provisioning steps.

iSCSI

The same “TDZ like” approach could be applied to iSCSI with a couple of minor changes to the way the iSNS is used. *Actually, first people would need to use the iSNS, but that’s another story...* The following gives an overview of one possible way this could be done.

A couple of things to take note of before I begin:

- The following example assumes that DHCP option 43 (vendor specific) is being used in a manner similar to what is [described here](#)
- To fully realize the benefits of this approach, a DHCP option specific to iSCSI-VLANs will be required
- The diagrams in this section show the iSNS and DHCP servers embedded in the switch chassis. I’ve only drawn it this way for convenience and they could just as easily be a centralized service running somewhere within the network. The point is this solution isn’t limited to single switch topologies.
- The switch will need to be administratively configured with:
 - at least two VLANs (e.g., 1 and 100) as shown in figure 1 (below)
 - The DHCP and iSNS servers will need to be enabled and accessible from the appropriate VLANs
 - The DHCP server will need to support something like the vendor specific option (option 43).
 - this option will need to return a list of VLANs that support iSCSI
- Each switch interface will need to be configured as a trunk port and the appropriate VLANs (e.g., 1 and 100) will need to be allowed on each of the interfaces

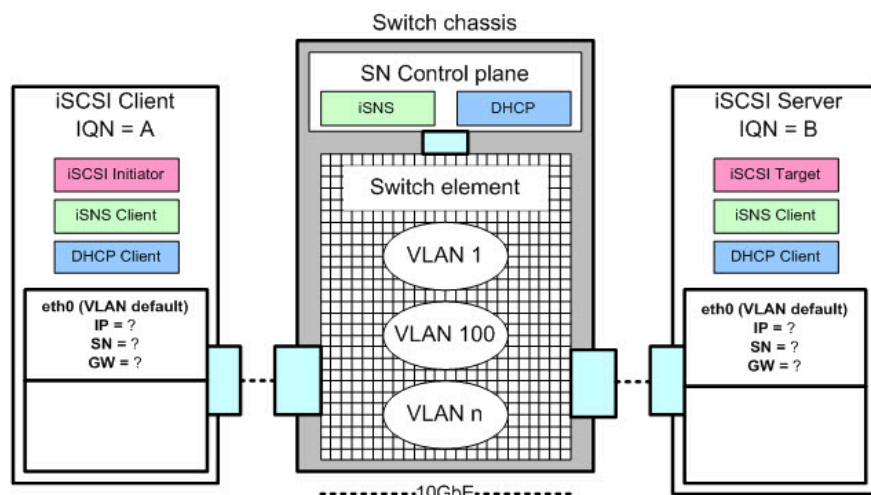


Figure 1 – Starting Configuration

iSCSI initiator configuration and registration

The iSCSI initiator interfaces will initialize as follows:

Note: An “enable iSCSI” configuration parameter may need to be provided to prevent iSCSI discovery from being performed on inappropriate interfaces. This overview assumes that such a configuration parameter has been provided and that eth0 (see figure 2) is configured to perform iSCSI Discovery.

1. During boot, link initialization, etc, the server containing the iSCSI client will discover the DHCP server. The discovery would be performed using the default VLAN and the server could request IP, subnet mask, gateway/router and a special parameter (option 43) that indicates the DHCP server should provide a list of VLANs that support iSCSI. *If you're familiar with FIP VLAN Discovery, then you'll understand where this idea came from.*

2. The DHCP server should respond with an IP Address, subnet mask, Gateway/Router, DNS (optional), and a list of VLANs that have been created for the purpose of supporting iSCSI (VLAN 100 in this case).

Note: interface eth0 will be used for the sake of this example but this process could be used on every “ethx” instance that has “iSCSI Discovery” enabled on it.

3. The iSCSI Initiator should configure the IP Address, subnet mask, default gateway and DNS of eth0 to match what was returned from the DHCP Server. (Normal DHCP)

4. A new “service” on the iSCSI Initiator should create one new interface for each iSCSI VLAN returned from the DHCP server. These interfaces will have the format of “ethx.y” where x is equal to the interface instance where the iSCSI VLANs were discovered and y is equal to the VLAN ID returned. For example, “eth0.100” as shown in figure 2.

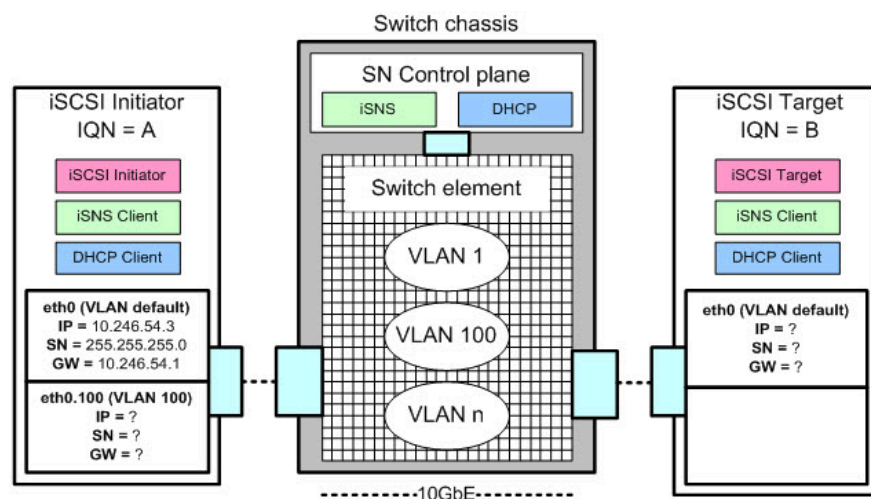


Figure 2- Eth0 configured

5. Once the iSCSI interfaces (e.g., eth0.100) have been created, DHCP discovery can be repeated on these interfaces with two exceptions:

- The iSCSI-VLAN option should not be specified. (since the iSCSI VLANs were already discovered)
- DHCP option 83 (DHCP option for iSNS) should be specified. (iSNS discovery)

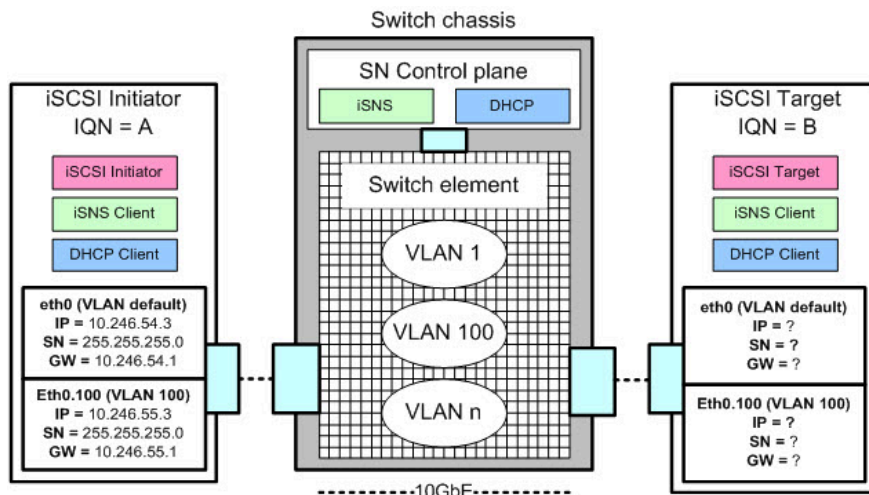


Figure 3 - After iSCSI interface creation

6. After the iSCSI interfaces have been configured with the information returned from the DHCP Server, the iSNS Client on the iSCSI Client should register with the iSNS. The information registered should include the IP Address and IQN. The iSNS client should also register for SCN.

7. After registering with the iSNS, the iSCSI initiator should query the iSNS for iSCSI Targets.

8. For each iSCSI target returned by the iSNS, the iSCSI initiator should perform login and SCSI Discovery (report LUNS, etc).

Note: At this point in our example, since the iSCSI Target is not yet registered with the iSNS, no iSCSI targets will be returned.

iSCSI Target configuration and registration

1. In order to facilitate the automation of the network provisioning process, each target will need to register information with the iSNS. This information should include:

- the same type of information that was registered by the Initiator; and
- will also include the list of iSCSI initiators that should have access to the iSCSI Target (if any). This will be accomplished through the use of the DDReg message.

2. Assuming the target has been configured to allow access to the iSCSI initiator, the initialization process shall be the same as defined for the iSCSI initiator with the following exceptions:

- The iSCSI Target will need to be configured as a management station and, as defined in RFC 4171:
 - register as a control node
 - register for management SCNs
 - register its own Login Control list using the DDReg message and listing the iSCSI name of each initiator to be registered in the target's DD
 - query for the discovery domain set (DDS) name/ID and then add their DD to that DDS.

Figure 4 shows the topology after the target has completed interface initialization.

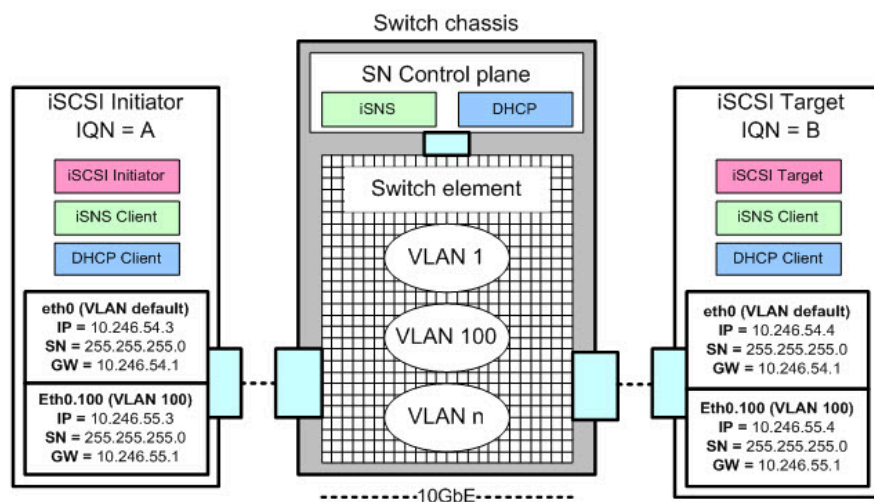


Figure 4 - After target interface initialization

iSCSI initiator discovery of targets after SCN

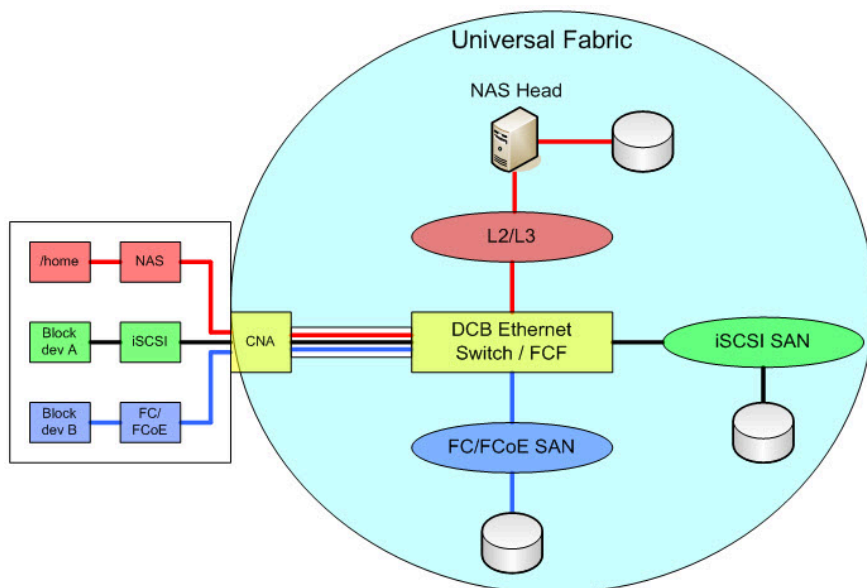
1. Once the iSCSI target has completed registration, the iSNS would transmit SCN to the iSCSI Initiator
2. Upon reception of SCN from the iSNS, the iSCSI initiator should query the iSNS for iSCSI Targets.
3. For each iSCSI target returned by the iSNS, perform login and SCSI Discovery.

NAS

I believe this same approach could be applied to NAS via something like Autofs (although I haven't completely worked through exactly how this would be done). It would require a centralized point for the NFS server to publish mount points / associated list of clients (users, IPs, subnets?) that were configured to have access these mount points. I believe this can be done via NIS or LDAP but as I've discovered with iSCSI, I wouldn't at all be surprised to find that a few changes would be required to existing implementations in order to get this to work. By the way, if anyone reading this is interested in thinking outside of the box with me and has NAS expertise, I'd really like to hear from you!

The Universal Fabric

If you consider the discovery automation functions that I've described over the past couple of sections, you've probably noticed a pattern. In each case I propose extensions to existing technologies/protocols that will allow for **incremental improvements** to each individual technology. While it's true that these enhancements require additional "up front" work to be done by the administrator, these additional steps will only need to be performed once. So let's assume for a minute that all of the infrastructure components have been configured and you no longer need to worry about those configuration steps on an ongoing basis. What you are left with is a simple way to provision and re-provision storage in one **protocol independent** way.



You would simply:

1. Connect to the storage array / NAS head and provision storage (LUNs / mount points)
2. The storage publishes this information in a protocol specific way:
 - For FC and FCoE, use [TDZ](#)
 - For iSCSI, use the iSNS as I described above.
 - For NAS, maybe NIS/LDAP would be appropriate.
3. The host could then automatically discover the storage that has been made available to it using any combination of the above three methods:
 - For FC and FCoE, an RSCN would prompt the host to perform discovery.
 - For iSCSI, SCN would prompt the host to perform discovery.
 - For NAS, Autofs could be set to poll the NIS/LDAP server (probably not a good idea) or the user could kick off a simple discovery script.

It's important to point out that, as is the case with TDZ, the presence of these "helper" functions would not impede normal discovery / provisioning if that is what you wanted to do with one or all of the protocols. In other words, this should be completely backward compatible with what is being done in your environment today because these functions merely **extend** what is already being done.

Thoughts?

Thanks for reading!

Posted at 07:34 PM | [Permalink](#)

Reblog (0)

Comments

You can follow this conversation by subscribing to the [comment feed](#) for this post.



Did you know that iSNS has a (maybe slightly flawed) fc model built in to it ? We've discussed some similar ideas over here more than once, and there is a lot of sense to what you say. Some small and subtle changes in a few places would be goodness. With servers and indeed storage being increasingly multiprotocol there is even more reason why this makes sense no matter the exact mechanisms you would use. IMHO the biggest issue here is that the politics within one standard body is tricky enough, but the politics across two or three standards bodies makes it hard to get some of this stuff done which I think is a shame.

Posted by: Simon Gordon | [05/19/2012 at 02:20 AM](#)



Thanks Simon, actually, I've decided to try and avoid the politics with the standards bodies for now and try something different this time. Stay tuned..

Posted by: [Erik Smith](#) | [05/19/2012 at 08:57 AM](#)

The comments to this entry are closed.

[Brass Tacks](#) [Powered by TypePad](#)