



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

[Main](#)

05/01/2014

Virtual Storage Networks - Part 3 (Multi-tenant capable topologies)

This is part three of a blog series that describes a new concept we're referring to as a [Virtual Storage Network](#).

In [part 1](#) of this series, I introduced the Virtual Storage Network or VSN as concept that enables the dynamic creation of storage service compositions. I also provided some background information and stated what we believe the requirements for multi-tenant storage are.

In [part 2](#), I provided additional detail about the multi-tenant storage requirements and made reference to a couple of topologies that could satisfy these requirements to one degree or another.

In this installment, I'll discuss these topologies and explain how many of today's infrastructure elements are unable to support multi-tenant isolation.

Before I go too much further, I need to clarify a few points.

1. In the previous two posts, I used the term "multi-tenancy" somewhat liberally to describe an environment consisting of a single set of resources that could be utilized by some number of tenants, but I never actually defined what I meant by "tenant". For the remainder of this post, assume that a tenant consists of

- one or more Virtual Machines,
- one or more Virtual Networks for LAN connectivity,
- some amount of storage capacity, and
- zero or more Virtual Storage Networks (VSNs) to connect the VMs to their storage capacity.

2. Outside of the VSN concept, all of the tenant elements described above exist today.

3. In the introduction above, I made the statement that many of today's infrastructure elements are unable to support "multi-tenant isolation". I did not say that today's infrastructure elements are



[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.

unable to support "multi-tenancy". This is a subtle yet important difference that will hopefully become a bit clearer as you read through this post.

4. Even though I believe many infrastructure elements are unable to multi-tenant isolation today, I am not, in any way, trying to imply that multi-tenant **isolation** is necessary or even desirable in all "multi-tenant" use cases. For example, I'm not trying to say that in a VDI environment we should be allocating a VSN to each VDI instance! But, it may eventually make sense to allow groups of instances (e.g., a tenant) to attach to a VSN that has been setup specifically for that group.
5. The final point is the difference between "multi-tenant" and "multi-instance". Multi-tenancy describes a model of resource sharing where multiple tenants are provided isolated access to a single instance of a resource. Multi-instance describes a model where each tenant is given access to their own separate instance of a particular resource. For more information, see [Wikipedia](#), this [gigaom blog post](#) or this [SugarCRM post](#).

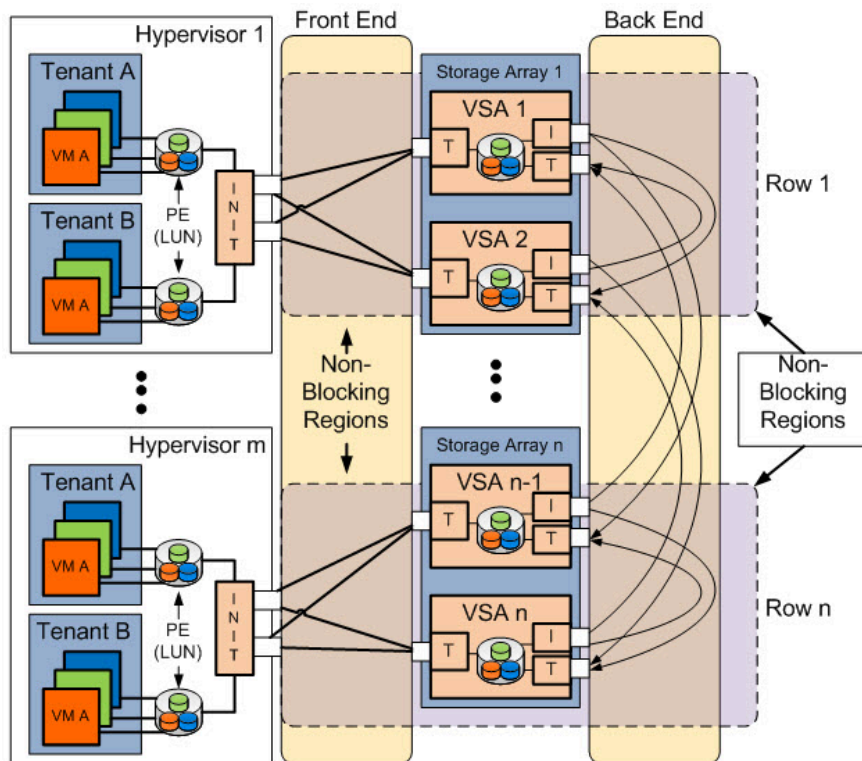
Satisfying Storage requirements

Recall that in [part 2](#), I provided a list of seven storage requirements and described why they were needed. For your reference, I've provided this list below and will explain how each topology shown meets or fails to meet these requirements.

1. Namespace isolation
2. Prevent noisy neighbor problems
3. Provide Bandwidth / IOPS / response time guarantees
4. Tenant traffic identification
5. Storage Network Service Insertion (e.g., encryption)
6. Authentication (e.g., CHAP)
7. Do not rely on Guest OS resident iSCSI initiators

Topology 1

The first topology I'll describe is shown below and has been logically sliced into vertical layers. From left to right these layers are compute, a front end network, a storage layer and then a back end network. It has also been physically sliced into horizontal layers with Row 1 on the top and Row n on the bottom. Each row could consist of some number of hypervisors or non-virtualized servers and one or more storage array instances. The front end and back end networks could utilize the same physical network.



Note that each hypervisor is hosting multiple tenants and each tenant has associated with it:

- one or more VMs;
- LAN connections (not shown for clarity); and
- in this case a single VVOL Protocol Endpoint (PE) essentially a SCSI Logical Unit (LU) containing multiple VVOLS. Each VVOL PE could be presented as a LU behind a tenant specific Target inside of a VSA.

For reasons that should become clear shortly and will be expanded upon in part 4 of the series; I've designated portions of the networks as "non-blocking regions" and have associated these regions with a Row. This non-blocking network could use FC, Ethernet or IB as a transport and make use of any protocol supported on the transport of choice.

Another point to keep in mind; although I'm showing VVOLS above, I'm not implying that only VMware could be used. This kind of topology could be used with any kind of hypervisor.

The Storage arrays have been logically segmented into Virtual Storage Appliances (VSA). For the sake of example, you could probably think of these as being similar to VNX Virtual Data Movers (VDMs). Each VSA provides a Target (T) interface that can be accessed by an Initiator on the Hypervisor. Since I'm not trying to limit the protocols that can be supported by this model, I feel it's important to point out that you could replace the "INIT" function on the host with "Client" and the "T" function on the storage with "Server" and it will not impact the model in general. Also note that each VSA has back end connections. These connections would be used for replication and mobility (refer to the [FAST Sideways](#) use case for information that describes how these connections could be used).

Finally, Topology 1 does not require any changes to existing storage stacks running in today's (existing) hypervisor or non-virtualized Operating Systems. That having been said, the ability to identify the storage belonging to a VM is a

compelling feature that needs to be accounted for in the design. In an ESX environment, this would require the use of something like VVOLs (when available). With other Hypervisors (e.g., KVM based) you could just deal with individual Storage Volumes (LUs) that are presented from the array. BTW, there are other options for identifying the storage belonging to a particular VM but I am not going to discuss them at this point.

Satisfying storage requirements with Topology 1

In this section I'll evaluate topology 1 to determine if it meets the requirements described in [Part 2](#) of this series.

Namespace Isolation

This requirement is partially met by Topology 1. The particular problem I focused on in the requirement definition was related to multiple OpenStack instances attempting to utilize the same array. Assuming that VSAs are being used and each VSA has an isolated name space, this use case could be supported by this kind of topology. The problem is, this topology does not address the [namespace](#) issue on the compute node and as you will see, this will cause problems with other requirements.

Prevent noisy neighbor problems

This requirement is not met by Topology 1. There's nothing to prevent acute network related noisy neighbor problems caused by one tenant from impacting another. However, the problem could probably be mitigated somewhat by using a non-blocking network topology between the compute and the storage and then being careful about how heavily you load each host and array interface. Also, chronic problems could be detected and be resolved via a mechanism like [FAST sideways](#).

Provide Bandwidth, IOPS and Response Time guarantees

This requirement is not met by Topology 1. There's only a single initiator per hypervisor and this means that something similar to the [IO blender](#) issue could still cause cross-tenant problems. I'm not saying the Bandwidth, IOPS and Response Time guarantees you can configure today won't work, they just do not provide this functionality at the correct level of granularity (e.g., on a per tenant basis) or from an end-to-end point of view.

Tenant Traffic Identification and Storage Network Service Insertion

This requirement could partially be met by Topology 1. Although VVOLs (or alternatively one LU per VM) will allow for per VM traffic identification, there is no way to identify a particular Tenant's storage traffic and differentiate it from another Tenant's storage traffic. This is because even if you had a map of every VM (via LUN) to Tenant relationship, in order for you to identify and perform actions on all tenant traffic, you would need to keep an enormous amount of state information (e.g., which FC exchanges belong to which tenant) and be able to act on this information in an end-to-end fashion. This is just not practical. An alternative would be to add a protocol specific header to each frame that would allow for this identification to take place and this would probably work well. However, since there is still only a single initiator that is shared across tenants, it doesn't address the problem completely.

Authentication (and encryption)

Since there is only a single data plane element (e.g., an Initiator) that is shared between tenants on the same Hypervisor, there is no way to perform per-tenant

data plane authentication.

Per-tenant encryption can be supported especially if you were to use application level encryption. However, if you wanted to encrypt a particular tenants data by using an encryption appliance, you would need to be able to identify all traffic from a particular tenant. As I described above, this could be done on a per LUN basis, but there are challenges with this approach.

Do not rely on Guest OS resident iSCSI initiators

Since the initiators used to access storage run on the Hypervisor and not within an individual VM, this requirement is met by topology 1.

Topology 1 summary

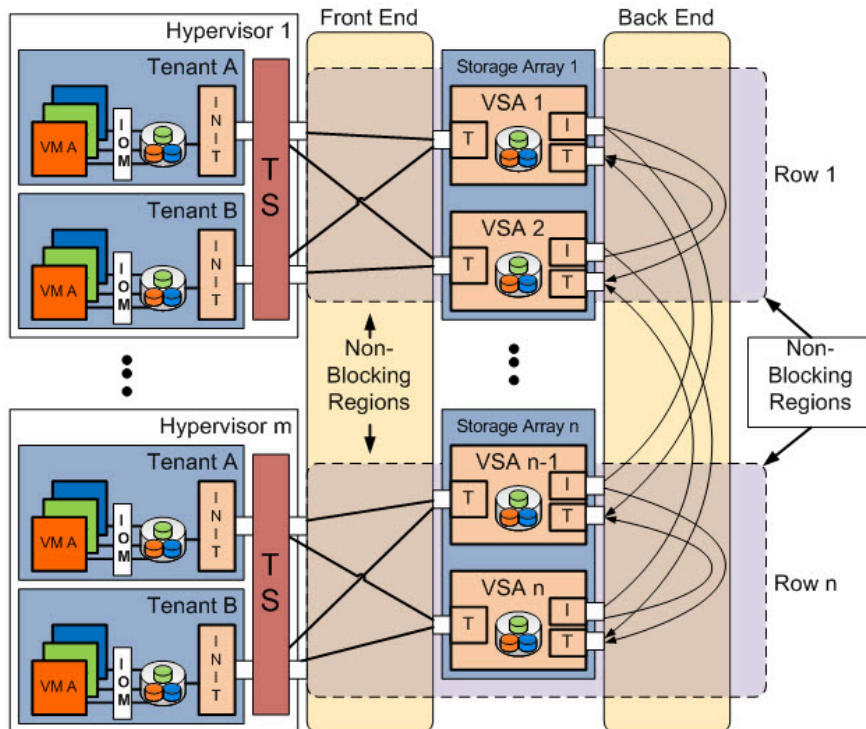
With the exception of VVOLs, topology 1 is being used by many Enterprise customers today and works completely fine for **many** Enterprise applications.

However, as I described in the [VSN overview post](#), Service Provider environments are fundamentally different and a much higher priority is placed on support for multi-tenancy in general. Since topology 1 is unable to meet most of the multi-tenant storage requirements identified, I think its usefulness in Service Provider environments is minimal. Next I'll explain a topology that could meet all of the multi-tenant storage requirements and that may prove to be useful for applications that require an increased level of isolation.

Topology 2

Topology 2 is similar to Topology 1 with a few “minor” changes that essentially allow each tenant to have a data plane personality that is separate from other tenants. I'll describe each of these changes and how they impact the specified requirement below; for now just know that these changes are:

1. one or more initiators per tenant,
2. a per tenant I/O Monitor (IOM) module, and
3. a Traffic Shaper entity that ensures a tenant does not consume more bandwidth than has been allocated to it.



Satisfying Storage requirements

As was done with topology 1, I'll go through each of the multi-tenant storage requirements and discuss how topology 2 does or does not meet each requirement.

Namespace Isolation

This requirement is satisfied by Topology 2. This of course assumes each iSCSI initiator resides in its own namespace (e.g., [network](#) and iSCSI namespaces). Each iSCSI initiator can be exposed via multiple network interfaces with each attached to a tenant specific Virtual Network. This Virtual Network could be a VLAN or another Virtual Network type (e.g., overlay based).

Prevent noisy neighbor problems

This requirement is theoretically met by Topology 2. I say theoretically because the introduction of a Traffic Shaper (TS) to the network edge could ensure that a given tenant does not consume more transmit bandwidth than has been allocated to it. There are a couple of challenges with this approach that would still need to be sorted out and the big one for me is the resource scheduler function. I mean consider a situation where you have allocated 5000 IOPS to a tenant. You would need to be able to share the IOPS fairly between all VMs whether or not all VMs are located on the same hypervisor. In any case, by performing the same TS function at the VSA and by using a non-blocking region of the network, we're probably about as close to a true end to end bandwidth "guarantee" as we're going to get for the foreseeable future. Note, if you decide to use a lossless protocol (e.g., FC, FCoE), congestion spreading could still cause cross-tenant issues. This would be a great area for T11's FC-NWSG (Fibre Channel New Work Study Group) to investigate.

Provide Bandwidth, IOPS and Response Time guarantees

This requirement is satisfied by topology 2. As I described above, I'd want to validate that the TS functionality solves the Bandwidth allocation issue. That

having been said, since each tenant utilizes a tenant specific iSCSI initiator, IOPS and Response Time could be allocated and monitored on a per tenant basis. The IOM would ensure that the effective IOPS and RT being experienced by the end users remain within tolerances. If this is not the case, the remediation steps I mentioned earlier could be used to resolve both acute and chronic IOPS and RT related issues.

Tenant Traffic Identification and Storage Network Service Insertion

This requirement is satisfied by topology 2. Again, each tenant utilizes a unique iSCSI initiator and as a result each tenant can be isolated onto a specific Virtual Network which could be rerouted through devices that provide per tenant storage services (e.g. in flight encryption). VVOLs (or alternatively one LU per VM) will still be needed because these allow for per VM actions to be performed on the array themselves.

Authentication (and encryption)

This requirement is satisfied by topology 2. Each tenant can be authenticated individually on the data plane. Also, as described above, since the tenant traffic can be easily identified it would be possible to pass it (and only it) through a storage service appliance (e.g. in flight encryption) without impacting other flows. Finally, as I mentioned earlier when I introduced the concept of an IOM, if the IOM included an encryption function, all data belonging to a tenant could be encrypted at the IOM before it leaves the tenant container.

Do not rely on Guest OS resident iSCSI initiators

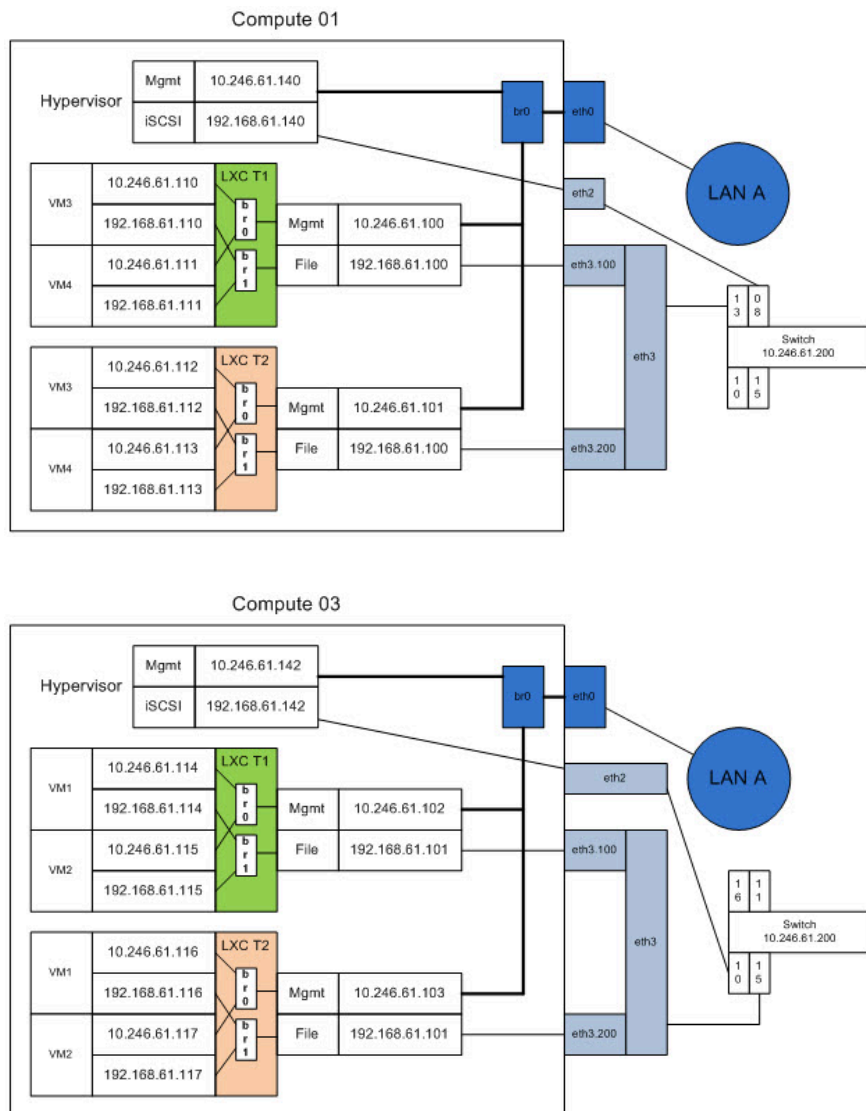
Since the initiators used to access storage run on the Hypervisor and not within an individual VM, this requirement is also met by topology 2.

Topology 2 summary

From a multi-tenant storage point of view, I think topology 2 has the potential to meet all of the requirements identified. That having been said, the majority of these concepts are still at an advanced development stage. If you're interested in utilizing any of these concepts or think we've missed something, I'd like to hear from you.

Topology 2 - Additional information

As I just mentioned, topology 2 is in what we'd refer to as the Advanced Development stage. To give you a better idea of what this means, I've included a couple of diagrams that illustrate the topology we used to validate the network namespace isolation requirement. Keep in mind, this is the topology that we're using for Proof of Concept (PoC) testing, this is not necessarily the topology that this functionality requires.



Some of the PoC topology specifics are:

- Ubuntu 13.10 was used for the “hypervisor”
- Two tenants (e.g., T1 and T2) were simulated using Linux containers (LXC)
 - Each container had its own management interface that was intended to be used by a Cloud or Infrastructure admin to configure tenant elements. Because this management interface was intended to be accessed remotely (via `br0` and the LAN) it needed to have a unique IP Address.
- Each container had File (storage) interface.
 - Note that the IP Address was the same on both File interfaces. These IPs didn’t need to be the same, but that was the point of the test; to validate namespace isolation was working for File interfaces in each container.
 - These File interfaces were not assigned to a particular VM, just to the container.
 - These File interfaces were connected to a bridge and then this bridge was connected to a particular

VLAN. Again, we could have (and have in the past) used an overlay Virtual Network instead of a VLAN.

- Two KVM based VMs we're instantiated inside of each container
 - This caused all kinds of issues with AppArmor. If you're going to try this be prepared to put all of the LXC related AppArmor profiles into complain mode.
 - Each VM had a public interface that was intended to be used for remote access.
 - Each VM also had a storage interface. We also could have used duplicate IP Addresses (e.g., T1-VM3 = 192.168.61.110 and T2-VM3 = 192.168.61.110)
- We had also planned to do some testing with iSCSI in this environment but we ran into an issue related to a lack of support for multiple network namespaces that prevented us from moving forward with it at this time. I believe someone else ran into [this bug](#) as well.

Conclusion

So this was a lot of information to try and cover in one post and then relate to the other posts in the series. With this in mind, the key points of the blog series up to this point are:

- Service Provider and Enterprise environments are fundamentally different (as discussed in the [VSN overview post](#)).
- Topology 1 is widely used today by many Enterprise customers but it does not provide data plane isolation between tenants and this is a feature which could be very useful in Service Provider environments.
- Topology 2 allows for data plane isolation between tenants, but a lot of work still needs to be done to work through the details... However, we're working on it and would value any inputs..
- Regardless of the need for data plane isolation, both types of customers are asking for the ability to automate storage provisioning tasks
- A big part of the storage provisioning process is establishing connectivity between storage consumers and producers.
- We would like to automate the establishment of connectivity between the storage entities to facilitate the creation of storage service compositions.

Next time... Connectivity Automation

In part 4 of this series I'll define what I mean by Connectivity Automation and describe how it can be applied to the topologies I introduced in this post.

Thanks for reading!

Posted at 04:40 PM | [Permalink](#)



Comments

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

The comments to this entry are closed.

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