


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

04/10/2014

Virtual Storage Networks - Part 2 (Requirements for multi-tenant storage)

In [part 1](#) of this series, I introduced the concept of a Virtual Storage Network, provided some background information and stated what we believe the requirements for multi-tenant storage are. In this post I'll dig into the details of those requirements.

Requirements for multi-tenant storage

As stated before, the following requirements were gathered by speaking with several of our customers who are interested in multi-tenancy. I'll describe each in detail and then in the next section, describe a couple of topologies that satisfy all of these requirements. The detailed explanations for each requirement will provide at least one specific example describing why the feature is needed. The example may or may not be the most realistic example possible, but it will be the example that I feel makes it easiest to describe why the requirement was asked for. Finally, keep in mind, the order does not indicate priority.

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

Namespace isolation

When I say “namespace”, I mean not only a [network namespace](#) although that certainly is a part of it. I'm also referring to things like iSCSI IQNs and in fact anything that can be used to uniquely identify a resource that is remotely connectable. One example where having more than one namespace would be useful is an environment that is using more than one “orchestration layer


[Erik Smith](#)
[Follow](#)

[ARCHIVES](#)

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

SEARCH

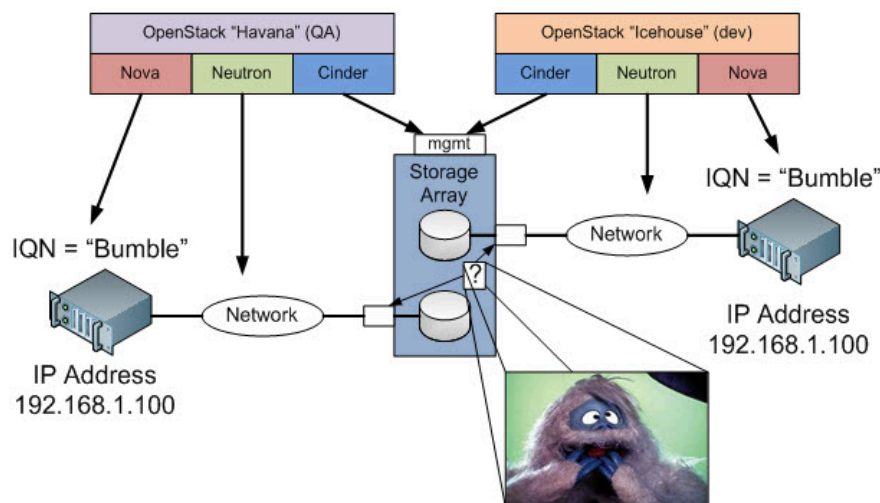
[CATEGORIES](#)

[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.

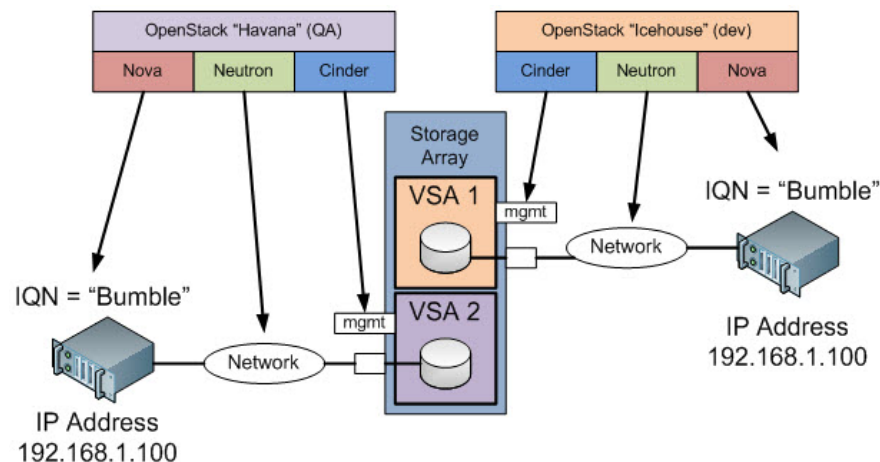
software" instance (e.g., two instances of OpenStack). In this case, let's assume we have two groups working with different OpenStack versions and they want to utilize the same Storage Array simultaneously and furthermore will connect to it via iSCSI. Maybe one group is working with the Havana release performing QA testing in preparation for rolling out a service, while the other team is performing dev/integration testing with the Icehouse release. For the sake of this example, let's assume the array only supports a single namespace.



Although this may seem like a reasonable configuration, there are a number of problems:

1. The Storage Array would need to be able to support multiple Cinder drivers simultaneously pulling from the same pool of physical resources.
2. The LUN Masking software on the array would need to be able to handle the case of a duplicate IQN being used without the risk of exposing one tenant's data to another. BTW, I'm not saying this is likely to happen, but it's possible.
3. The duplicate IP Addresses used by both tenants would probably cause the biggest problem in this case (I mean, just look what it's doing to "Bumble"). Due to the way that routing and forwarding works, the array would be unable to determine which network interface to use when responding to an IO.

As shown below, one possible solution to these problems would be to provide isolated namespaces for each OpenStack instance.

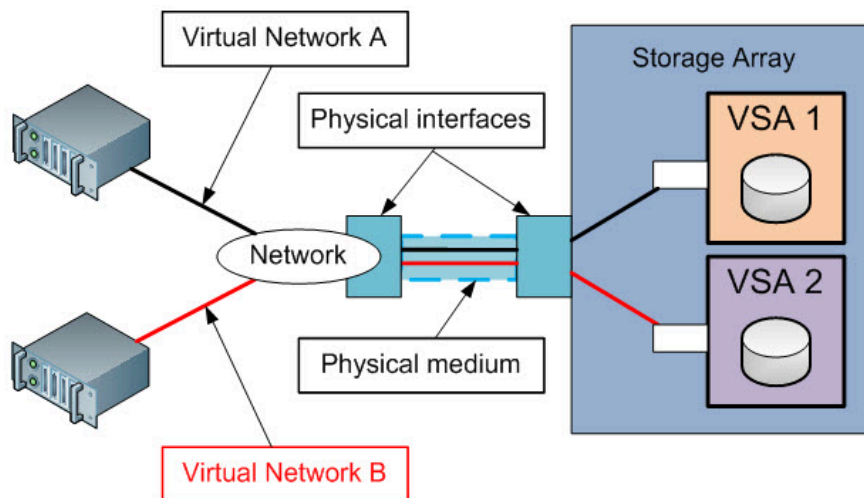


This could be achieved by utilizing a Virtual Storage Array (or something like it). If you're unfamiliar with this concept, you can think of each VSA as a Virtual Machine that provides storage services and contains its own network namespace. As far as each VSA would be concerned it would be under the control of a single Cinder driver and would not see duplicate IP Addresses (as long as the Networks shown above are separate). Also, since provisioning is done on a per VSA basis, there would be no issue with a duplicate (spoofed) IQN getting access to volumes that they shouldn't. Today, EMC supports the concept of a Virtual Data Mover (VDM) on the VNX platform but I would expect to see support for this feature expanded as it makes sense to do so. For now let's just assume that the concept of a VSA is useful in certain scenarios and that they can be dynamically instantiated as needed (like a VM).

Prevent noisy neighbor problems

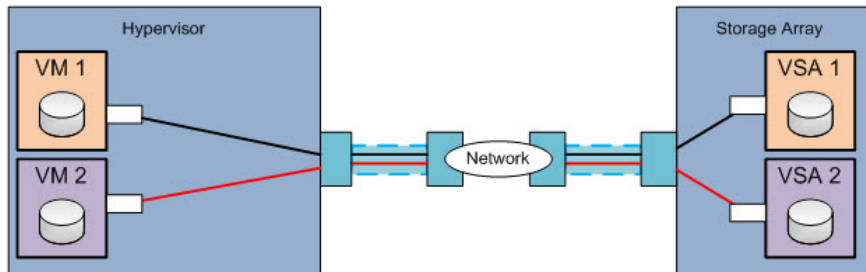
When I use the term “noisy neighbor”, I'm referring to the ability of one network end device to negatively impact the effective Quality of Service (QoS) being experienced by another end device. There are also many other types of noisy neighbor issues (e.g., spindle contention) that can be experienced by tenants that are sharing the same physical resource (e.g., physical disk). Although these other types of noisy neighbor problems are serious ones that need to be addressed, they are not strictly related to connectivity, so I'm not going to attempt to discuss them here. I will say that we have a group of very smart people working on physical resource sharing and they are continually enhancing our products to ensure the resources that are allocated are actually delivered to the end user.

In terms of a noisy neighbor as it applies to connectivity, using the previous diagram it's probably not immediately apparent that there could be an issue. However, if I redraw things slightly it should become a bit easier to understand.



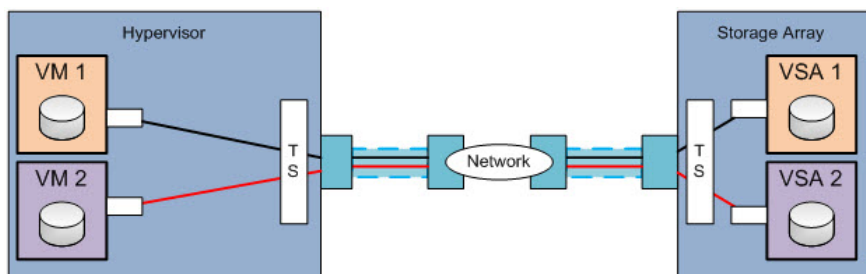
The diagram shows both VSA instances are being connected to by a different physical host via a different Virtual Network instance. This Virtual Network Instance could be a VLAN, a VXLAN or both VLAN and VXLAN. I will be providing much more detail about this concept later in this series. The point relevant to noisy neighbor prevention is that both VSAs share a common physical interface. If this interface were using a lossless protocol such as FC, or a protocol that used DCB (FCoE and Lossless iSCSI), congestion in one VSA would impact traffic to the other VSA if all of the VSAs were using the same priority or more specifically CoS. This is not only theoretically possible (since there are only 8

ethernet CoS), but in practice many switches only support a single lossless CoS, so it's likely that you'd encounter this situation. If you're interested in congestion spreading in a lossless network, check out the Congestion and backpressure section of the [Networked Storage Concepts and Protocols techbook](#). In the case when traditional iSCSI, File and even Object protocols are used, there's nothing to prevent spikes in traffic in one Virtual Network from impacting the other. This problem can be exacerbated in a Virtual Server environment as shown below.



Although not explicitly shown in the diagram, the configuration above would require that the storage protocol in use be terminated in the VM. While this is possible with iSCSI, it violates one of the requirements I'll talk about later and in general is not done all that frequently anyway. It's much more likely that someone would want to use a File or Object based protocol in this kind of configuration. In any case, you can imagine that since both VMs are using the same physical connection and there's no arbitration of network resource allocation being done, it's possible for each end device to impact the effective QoS being observed by any of the others.

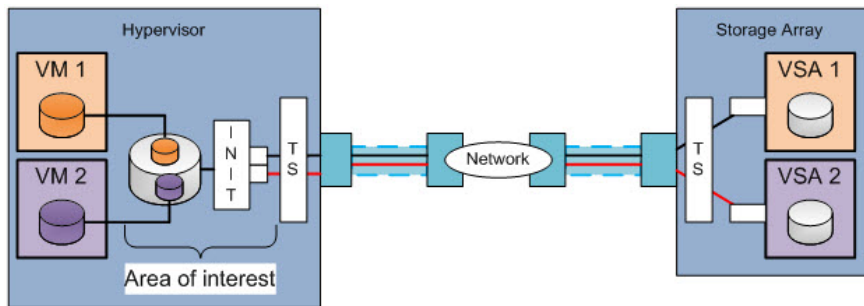
One theoretical solution to this problem would be to implement some kind of traffic shaping (TS) entity at the network edge as shown below. The problem is, for reasons that are outside the scope of this post, guaranteeing end-to-end bandwidth for an arbitrary number of end points across an arbitrary network topology is something that has proven to be an elusive feature to develop.



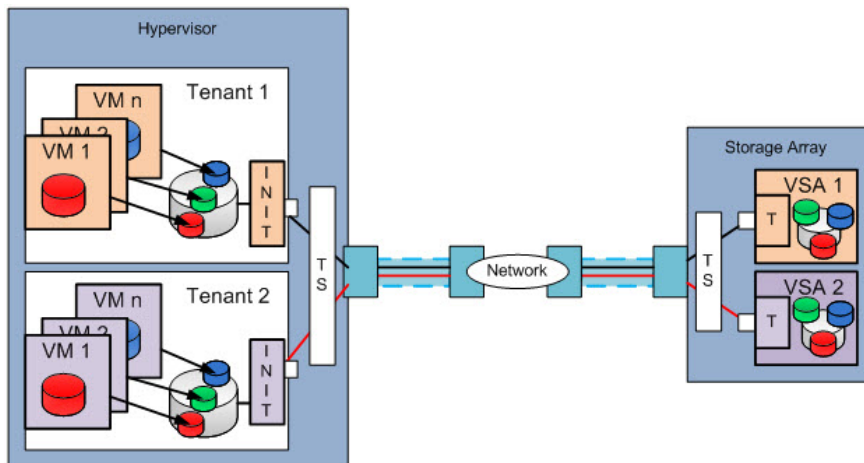
All of that in mind, let's assume that it will be possible to use something like this approach to limit the amount of bandwidth consumed by an end device. Although I do realize that this not the same thing as bandwidth guarantee, I'm only suggesting it because I know where I'm going with this line of reasoning. I'll just beg for your indulgence until I get to the topologies portion of this series.

As I mentioned earlier, many end users don't want to make use of iSCSI initiators that reside inside of the Guest (VM) and this further complicates things for us because of the way iSCSI traffic originates from a hypervisor such as ESX. As shown below, the problem is unless you're using an iSCSI HBA, there's a single iSCSI Initiator instance and this has some interesting side effects. The most important one from a connectivity point of view is that all iSCSI storage traffic appears to originate from a single iSCSI initiator. Of course

this single initiator can be presented out of multiple network interfaces but short of creating a data store for each VM and then making sure the LUNs in the datastore are only visible via a specific NIC, there's currently no way to associate a particular VM and its associated IO with a particular network interface. It seems like every time I point out this issue to someone, they start by asking "aren't VVOLS are going to solve this problem?" and the answer is NO! Again shown in the diagram below I've added some smaller disk shaped objects that are intended to represent VVOLS. The point being VVOLS are a SCSI level entity that sit one level above iSCSI in the stack, so VVOLS allow for the identification of VMs but all IOs for a group of VVOLS that are being accessed at the same protocol endpoint will still experience the IO blender issue.



An architecture that resolves the IO blender issue is shown below. Note that there are multiple tenants each with their own iSCSI Initiator. Traffic could be shaped on a per Initiator basis such that each tenant is allowed to consume a fixed amount of bandwidth. Again, this is not a guarantee from the network, just a guarantee that on a per hypervisor basis each tenant can only consume a certain amount of network bandwidth. VVols could still be used in this configuration to allow for per VM storage actions to be performed.



So the astute reader will note that the vast majority of what is described above is not supported today and they would be right. However, I will tell you that with the exception of VVols, we've set something like this up in the lab using LXC containers as the tenant entities and KVM VMs for the VMs. We did run into a number of problems but that's a story for another post. BTW, as I was writing this I noticed a [blog post](#) and thought the timing was funny given the fact that I've just essentially alluded to nested Hypervisors. :-)

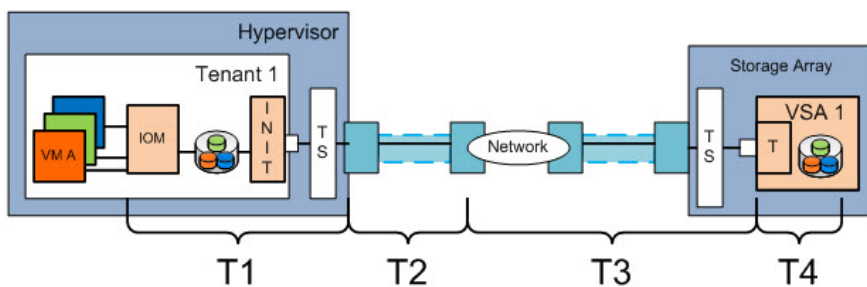
All joking aside, the majority of the "tenant box" above could be implemented as management abstractions, the only piece required to eliminate the cross tenant IO blender effect is one initiator per tenant. Keep in mind that each

initiator would probably also want to exist in its own namespace for the reasons that I described in the “namespace isolation” section above.

Provide Bandwidth /IOPS / response time guarantees

Above I described how a “traffic shaper” could enforce bandwidth utilization limits on a per tenant per server basis; I’ll come back to this later when I describe a couple of topologies that could meet all of the requirements. In this section I will briefly discuss providing IOPS and Response Time (RT) guarantees.

When my colleagues from the array teams talk about IOPS and RT, it’s typical for them to think of this in terms of the T4 (T = Time) segment of the diagram shown below. This is because they know exactly when the target (labeled as T in the VSA) receives the IO, how long it takes them to complete it and how many times they do this on a per initiator per second basis. Also when you provision storage and assign IOPS and RT performance parameters on the array, this is really the only way you can think about it. This is because although the Array knows exactly how many IOPS it’s performed and whether or not it is capable of satisfying the IOPS allocated to each initiator, it has no way to determine (let alone alter) how diagram segments T1, T2 and T3 are impacting the effective RT at the VM or how other factors (e.g., congestion in the network) are impacting the ability of a given initiator to perform the number of IOPS allocated to it.



Some networking vendors, especially Brocade with their Fabric Vision concept, are trying to address this problem by adding functionality that allows them to monitor for and report congestion in segments T2,T3 and to a limited extent T4. While this certainly represents a huge step in the right direction, I don’t believe it addresses segment T1. In addition, although I know there are a number of customers using things like [Storage IO Control \(SIOC\)](#) with ESX, I don’t think it completely addresses this problem either.

In my opinion, in order to address the problem from end-to-end (T1-T4), an additional block of functionality will need to be added and I’ve labeled it as “IOM” in the diagram above. As I’ll describe in subsequent sections, the IOM (I/O Manager) would ideally contain three functions:

1. IOPS and Response Time (RT) Monitoring
2. Remediation for IOPS and Response Time related issues
3. Encryption

For now, we’ll just focus on the first two and discuss encryption later.

Monitoring for Response Time was briefly described above and is fairly straight forward. Essentially the IOM would put a timestamp on each I/O as it “enters” the module. When the I/O completes the total IO completion time can be calculated.

- If the I/O completed within the allocated time no additional action would be required but the completion time value could be stored for some period of time. Perhaps even a rolled up average could be stored for longer periods of time.
- If the I/O did not complete within the allocated time, this condition could be flagged. If enough of these conditions occurred in a pre-defined period of time, the remediation functionality could be invoked. I'll provide more detail on this below.

Monitoring for IOPS is a bit more convoluted because although you could set a minimum IOPS threshold, you would need to be able to distinguish between the following two cases:

1. a VM would like to perform additional I/O but it cannot for some resource related reason, or
2. it just doesn't have any more I/O to perform that second.

The solution to the above would almost certainly involve monitoring I/O queues and would be implementation specific.

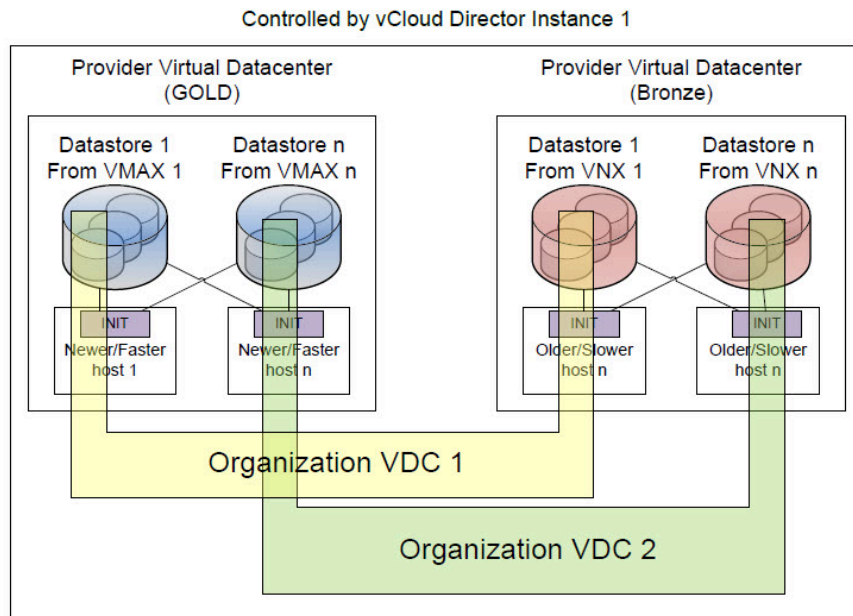
Once you've identified an IOPS or Response Time related issue, the remediation functionality would need to determine the correct action to take to resolve the issue. The action could be taken automatically and this would be the expected action for severe issues. One example would be a path going down (loss of a link) and the action that something like PowerPath would take to resolve it.

The action could also be simply to notify the administrator and this might be a better remediation step for chronic issues, especially if the remediation step involves relocating the storage volume to another array (e.g., [FAST Sideways](#)).

Tenant traffic identification

The basic idea behind this one is to allow for the identification of frames or packets that belong to a given tenant and to be able to perform this identification anywhere in the infrastructure. At least one Customer was interested in the feature for a security related reason.

For better or worse, when I think of a tenant, I tend to think in terms of vCloud Director and its concept of an Organization VDC, see below for my graphical interpretation.



If we use this model and we decide that we want the ability to identify a tenant's traffic between the Initiator (labeled INIT) and the storage (the blue or red "soup can" shape above the INIT), then we would need to isolate each tenant onto a separate physical host. Again, as I described above, the reason for this has to do with the relationship of an iSCSI initiator to a ESX host. This relationship also has implications to Storage Network Service Insertion as I describe below.

Storage Network Service Insertion

Storage Network Service Insertion is similar to [Network Service Insertion](#). Both Network Service Insertion and Storage Network Service Insertion support:

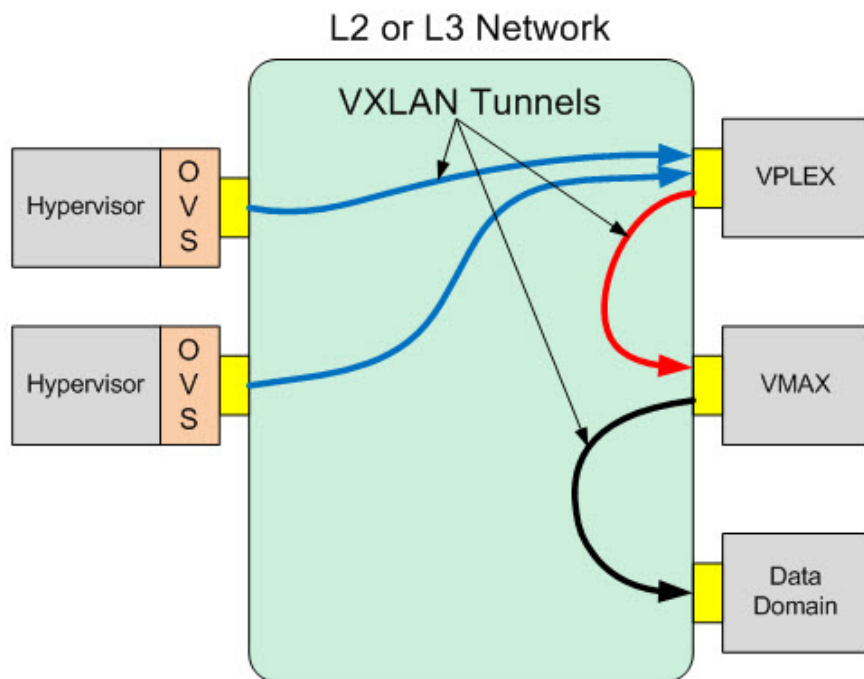
- dynamic service insertion (e.g., inserting a new service into an existing flow); and
- service composition (e.g., building a network service and then deploying it for use).

Examples of dynamic Storage Network Service Insertion could be:

- the addition of an inflight encryption appliance;
- the redirection of a flow to a security appliance for a closer look at access patterns, etc; or
- the [FAST Sideways](#) use case. Essentially, you could migrate a storage volume to a different storage appliance and take advantage of features supported by that appliance.

The above use cases require that tenant flows be isolatable and this is one of the reasons that we think one initiator per tenant is interesting.

An example of Storage Service composition, is shown below.



Typically, I like to think about Service Composition in terms of a user requesting a specific amount of capacity with a set of desired attributes. I won't go into details about the services themselves and how they relate to one another other than to say that in this case:

- the Initiator on the Hypervisor would access a Virtual Volume on the VPLEX
- The Virtual Volume would in reality consist of multiple services each being provided as follows:
 - VPLEX would provide the "distributed cache coherency" and "addressability" services
 - VMAX would provide the "storage capacity" service
 - Data Domain would provide the "dedup" service

The important piece to consider here is that these Storage Service Compositions could be dynamically created as needed without the need for pre-provisioning or manually configuring connectivity.

Authentication (and Encryption)

The customers we spoke with not only wanted the ability to authenticate the identity of an Initiator and a Target. As we dug into this requirement some more it turned out at least one of them also wanted the ability to provide Encryption as a Service (EaaS) to their tenants. Originally when we were talking about EaaS, we were thinking about it in terms of a service that you could insert between the initiator and a trusted network device. Once encrypted, the flow could take any path to reach another trusted portion of a network then be decrypted and sent to its Storage Target. This seemed to work for many use cases. However, eyes really seemed to light up when we asked if it would make sense to put this functionality inside of the IOM block I described above.

The basic idea is that the tenant would purchase the encryption service from the service provider and part of the provisioning process would involve the IOM interacting with a key manager on behalf of the tenant perhaps using [KMIP](#). The

idea is to provide tenant level encryption without requiring tenant involvement in the key management process.

Do not require Guest resident iSCSI Initiators

At least one of the customers we spoke with had supportability and security concerns with an approach that relied on iSCSI initiators being installed in the Guest OS (VM).

As it turns out, we had a scalability concern with this approach especially if every VM (up to 4000 in a rack) were to utilize a common storage array. The problem is each login consumes system resources and potentially creates management complexity. Because of this problem and due to the fact that at least one customer didn't like the approach, we decided to focused on per tenant iSCSI Initiators (when iSCSI is required)

Next time...

OK, well that pretty much sums up the description of the requirements. In [part 3](#) I'll propopse a couple of topologies that either partially or completely satisfy the requirements described above.

Thanks for reading!

Posted at 08:20 AM | [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](#)