



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

[« Zero Touch Infrastructure Provisioning \(ZTIP\)](#) | [Main](#) | [The power of Layers](#)

»

12/04/2017

ZTIP - Part 3: Provision and Transport

This is part 3 of a multi-part series that describes a “side-project” a small team of us (Jean Pierre (JP), Alan Rajapa and Massarrah Tannous and I) have been working on for the past couple of years.

In [part 1](#), I introduced an early version of this concept referred to as “Zero Touch **Storage** Provisioning” and described how it could be used to automate storage provisioning in with OpenStack.

In [part 2](#), I described why we changed the name to “Zero Touch **Infrastructure** Provisioning” (ZTIP), provided a [link to a video](#) that provides context about the project, introduced the IaaS overlay and underlay concepts, and then started to describe the details behind the IaaS underlay.

In this post I’ll continue my explanation of the IaaS underlay by starting with the Bootstrap (Provision) layer and working my way up from there.



[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.

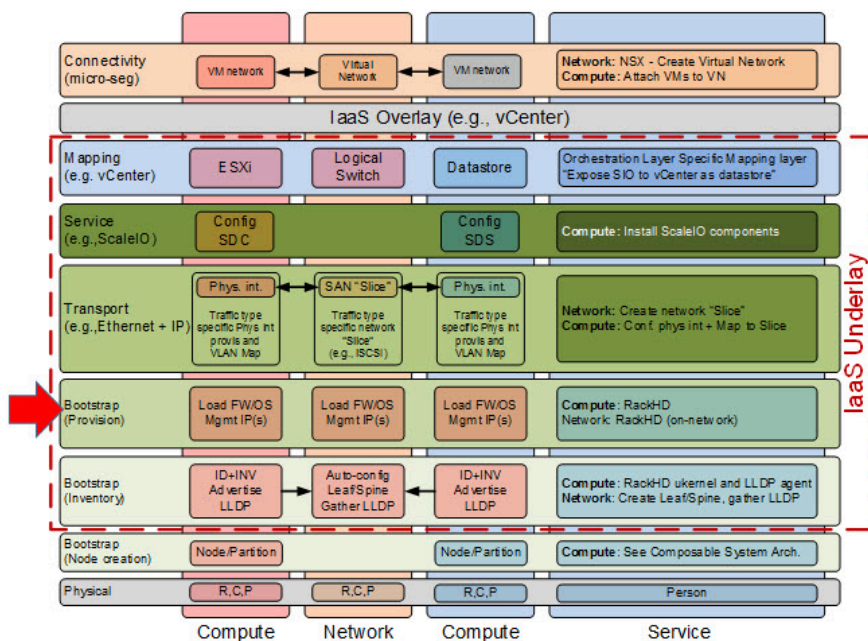


Figure 1 – IaaS Layers

Bootstrap (Provision)

In the last step of the previous blog post we provided a basic configuration for the network and then inventoried the servers using RackHD. The result of this discovery process was the realization that we have three cabinets each containing a different node (compute) type (i.e., GPU capable, Storage heavy and Compute heavy). See the diagram below.

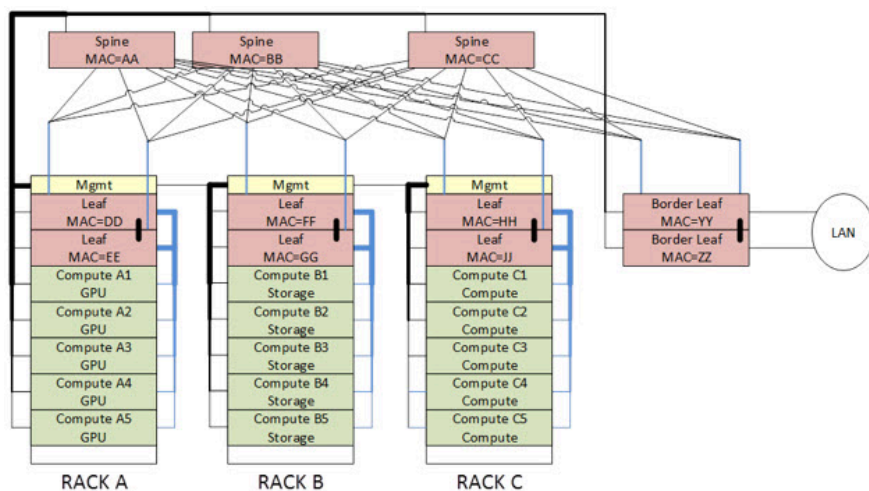


Figure 2 – Inventoried set of Racks.

Since we now know the Node type, we could:

1. Leave the nodes powered off and only power them up / load the OS when capacity needs require it. In this case only the iDRAC (BMC) interface would have an IP Address associated with it.
2. Load a default OS associated with that node type. In this case we would also want to assign the management IP to the management interface on the node once the OS is loaded.
3. Wait for the end user or a higher level orchestration layer to provide direction.

In this case we're going to assume an end user would like to configure a subset of the above nodes and we'll refer to that subset as a Logical System. More on

this shortly.

An aside: I used to refer to the concept of a Logical System as a “Hardware Pool”, but switched to the term Logical System after watching the [VxRack Manager prototype video](#). I’m not exactly sure what happened to VxRack Manager, but I do think they were digging in the right area. Perhaps the [Symphony project](#) will eventually support this kind of functionality.

I should also mention that I see a Logical System as being a fairly low level way to support multi-tenancy. In other words, one or more Logical Systems could be allocated to a particular tenant for isolation purposes.

Transport Configuration (e.g., Ethernet + IP)

Network configuration (Phase 2): Network Slice Creation

Logical System overview

For the sake of this example, we will assume that the end user has decided to create a Logical System that will require RDMA over Converged Ethernet (RoCE) to support “training” for the creation of a model to support real-time data analytics. We will also assume that the user has specified the size and location of a data set that they would like to work with and that based on the size of the data set and the operation to be performed on it, someone or something (e.g., the user or some intelligent workload placement algorithm) decides that it will require 4 GPUs, 100 TB of storage capacity and 2 dense compute nodes. We will also need connectivity to the data set (to be ingested) as well as to the Customers LAN. Assuming each GPU capable node has 2 GPUs and each storage dense node has 25 TB of storage capacity, this could result in the following nodes being selected and included into a GPUaaS Logical System.

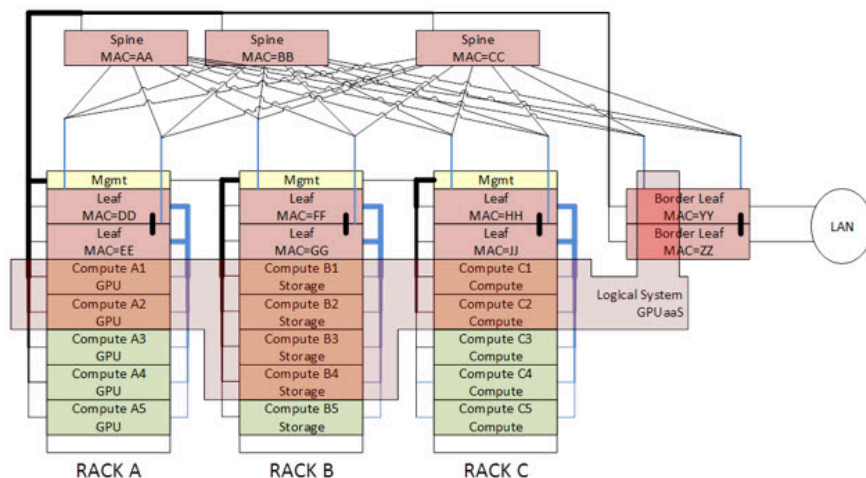


Figure 3 - GPUaaS Logical System

From a network connectivity point of view, we need several different types of networks to support the connectivity and isolation requirements of this GPUaaS Logical System.

- Console
 - **Provides:** Access to Compute Console (iDRAC)
 - **Instantiation:** Tenant (i.e., Logical System) specific Management LAN on Management Network
 - **Note:** Once a compute resource has been dedicated to a particular logical system, its BMC interface will need to be isolated from all other system resources that do not belong

to the same Logical System. This could be accomplished via ACLs on the management plane switches (e.g., Dell N3000 series)

- Management
 - **Provides:** Access to the physical management interface on Compute instances.
 - **Instantiation:** Tenant (i.e., Logical System) specific Management LAN on Management Network could be the same network as the Console network as long as these interfaces do not need to be remotely accessible.
- LAN:
 - **Provides:** VM / Container to world connectivity.
 - **Instantiation:** Tenant specific LAN Overlay Virtual Network on the “Default slice” of the Data Network. (Note: more information about the concept of a slice will be provided below)
- ScaleIO 1:
 - **Provides:** Dedicated/Isolated connectivity between compute nodes for ScaleIO traffic.
 - **Instantiation:** A “ScaleIO 1” network slice on Data Network with each interface ideally being allocated a minimum bandwidth.
- ScaleIO 2:
 - **Provides:** Dedicated/Isolated connectivity between compute nodes for ScaleIO traffic.
 - **Instantiation:** A “ScaleIO 2” network slice on Data Network with each interface ideally being allocated a minimum bandwidth.
- Ingest:
 - **Provides:** Connectivity to the location of the data source.
 - **Instantiation:** A “Data Ingest” network slice on Data Network with each interface allocated a minimum bandwidth.
- GPU:
 - **Provides:** RDMA connectivity between the GPU and the Storage nodes.
 - **Instantiation:** A “RoCE” network slice on Data Network with each interface allocated a minimum bandwidth and the network itself configured to support RoCE (i.e., PFC and ECN).

Network Slices

Some of the networks described above are instantiated as network “slices”. Before we dig into the reason for this, let’s start with a definition.

A network “slice” is an Underlay Virtual Network that has been instantiated for the purposes of logically isolating a portion of a physical network topology.

Each slice can have specific attributes assigned to it (e.g., DCB) which, for example, would allow for the transport of a protocol that requires losslessness (e.g., RoCE). A slice typically consists of at least two VLANs connected via an L3 (routed) portion of the network. The characteristics of a slice (e.g., losslessness) are expected to span the Network Topology from ingress to egress.

A slice is different from an Overlay Virtual Network (e.g., VXLAN) because slices are instantiated on, and require special handling from, network hardware to support the requirements of the traffic that will be transported (e.g., RoCE). Because each traffic class can consume finite network hardware resources (e.g., HW queues), we believe that it would be necessary to reuse slices across different tenants but in order to maintain tenant isolation, it will probably require the use of something like an Overlay Virtual Network (e.g., VXLAN).

A final note on the concept of a slice. I think it's fair to say that this concept was, at the very least, inspired by the work that a group of folks at Jeda Networks did with FCoE a few years back. The key difference is their approach was used to support FCoE traffic and we're thinking of using the concept in a more general purpose kind of way.

Just to highlight the difference between the underlay (e.g., a slice) and the overlay (e.g., an Overlay Virtual Network), I'll provide an example from some related work (below). Please note, although the following diagram provides an illustration of the slices concept using iSCSI over RoCE, the same principles could eventually be applied to other protocols such as NVMe over RoCE.

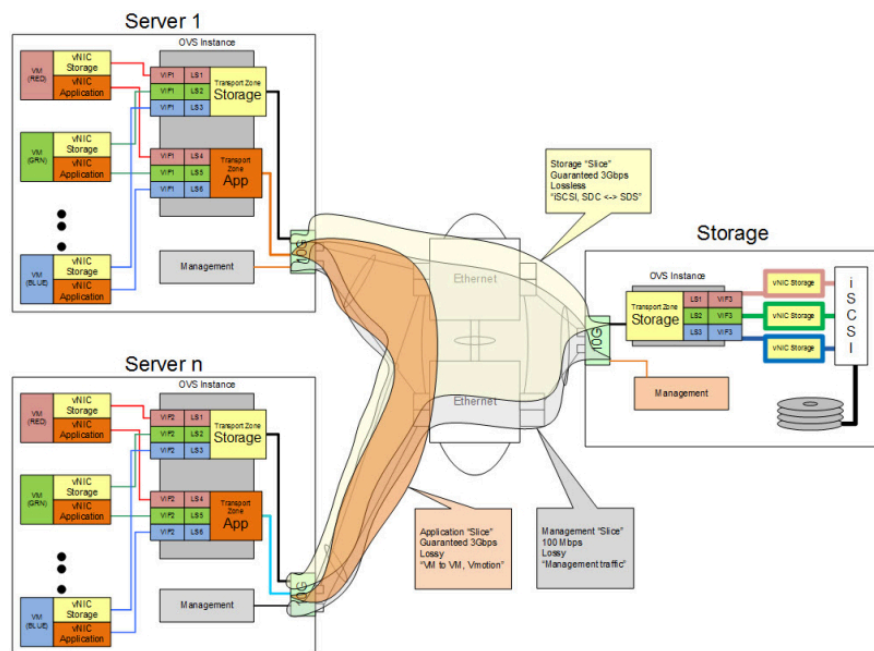


Figure 4 - Network Slices

In order to allow multiple tenants (e.g., Red, Green and Blue) to share each slice would probably require the use of an Overlay Virtual Network. These Overlay Virtual Networks are shown below as colored lines and could be configured at any point after the Slices have been created.

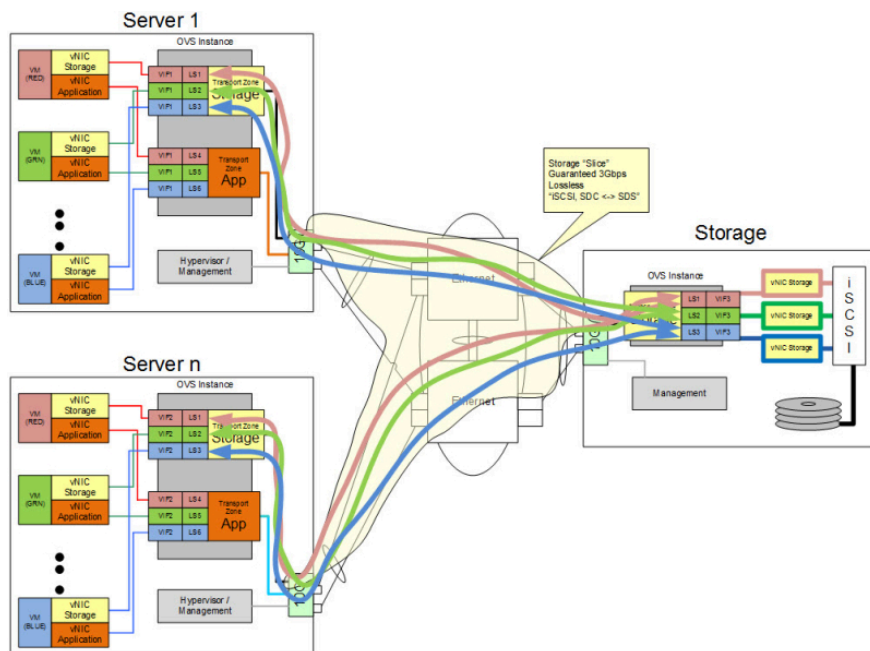


Figure 5 - Storage Slice with tenant specific overlays

An example of how this concept might be applied to a physical network is provided below.

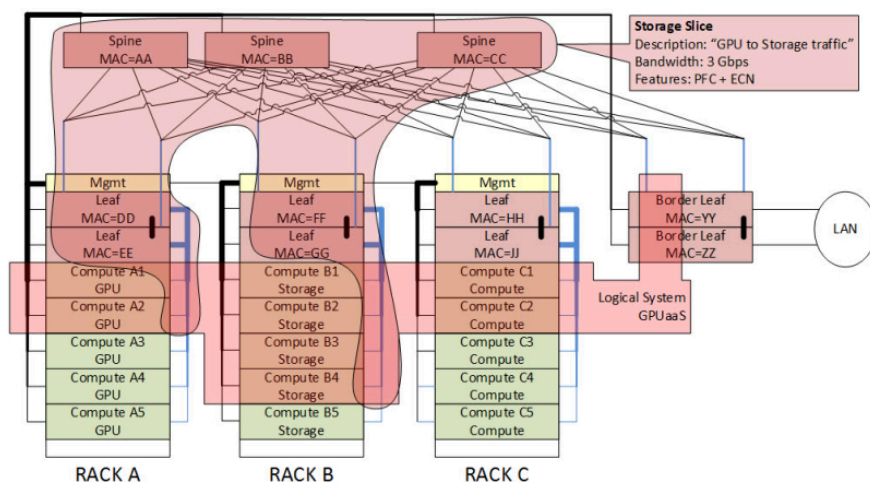


Figure 6 - Storage Slice instantiated on network

As shown above the storage slice connects the Compute nodes containing GPUs in Rack A to the Storage dense nodes in Rack B. This slice will ensure that there's at least 3 Gbps of bandwidth available per interface (oversubscription could impact this) and also ensure that the links that transport the traffic support PFC and ECN.

There is so much additional detail I could provide about the slices + OVN concept shown above. Areas of interest for me have been things like how to address the complexity associated with applying a BW limit at the slice layer and then further subdividing this BW at the OVN layer. It's an area that's ready for some serious innovation but again, you could probably find a networking vendor who could simplify this problem for you.

If you'd like to see some related work that we've previously done in this space, see the "[An Introduction to Virtual Storage Networks](#)" blog post series.

Today, there are many different approaches that could be used to instantiate a slice, but in this case we will assume that something like the steps described by

Mellanox in “[How To Configure Mellanox Spectrum Switch for Lossless RoCE](#)” have been performed.

Compute Node Physical Interface Configuration

To allow a compute node to access a specific slice, you would need to configure the switch interface to allow access to the VLAN associated with the slice and then configure the compute node interface to access the appropriate VLAN. A couple of points about this:

1. I am assuming that the network will be configured with the L3 boundary at the ToR (Leaf) switch and this may not always be the case.
2. In order to know what switch interfaces you’ll to configure, you will need to know where each server interface (that is part of this logical system) is attached and what each interface will be used for. This is one of those causality dilemma’s I described earlier and would be solved by a topology service.

Service (e.g., ScaleIO) configuration

Once the Transport layer (e.g., slice) has been configured and you have optionally configured OVN, you can start deploying and configuring the services that will use them. One example would be ScaleIO and we described how you could do this in the [ZTIP demo](#). Apparently, based on the [information that has already been made publicly available](#), you will also soon be able to use AMS to perform the same steps.

In the next blog post I’ll provide another example that uses iSCSI and perhaps an NVMe over RoCE example after that. :-)

Thanks for reading!

Posted at 10:09 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](#)