

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

12/01/2017

Zero Touch Infrastructure Provisioning (ZTIP)

This is part 2 of a multi-part series that describes a “side-project” a small team of us (Jean Pierre (JP), Alan Rajapa and Massarraah Tannous and I) have been working on for the past couple of years.

You’ll note that in [part 1](#) of the series, I was referring to a similar concept as “Zero Touch **Storage** Provisioning”. The reason for the name change was that along the way, we figured out that we were trying to provision WAAAY more than just storage, so we changed the name to “Zero Touch **Infrastructure** Provisioning” (ZTIP).

Before we begin, if you’d like to get an idea of the overall concept, as well as see a snapshot of where we were in the journey about 18 months ago, please see [this video](#) that was put together by Massarraah and JP. We (ok, I) like quirky names, so please do not hold the name of our controller “Orchestration System for Infrastructure Management” (OSIM) against them. :-)

Underlays and overlays

Our work rests on the idea that Infrastructure as a Service (IaaS) can be logically broken down into at least 2 layers; the IaaS overlay and the IaaS underlay.

[Erik Smith](#)

[ARCHIVES](#)

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

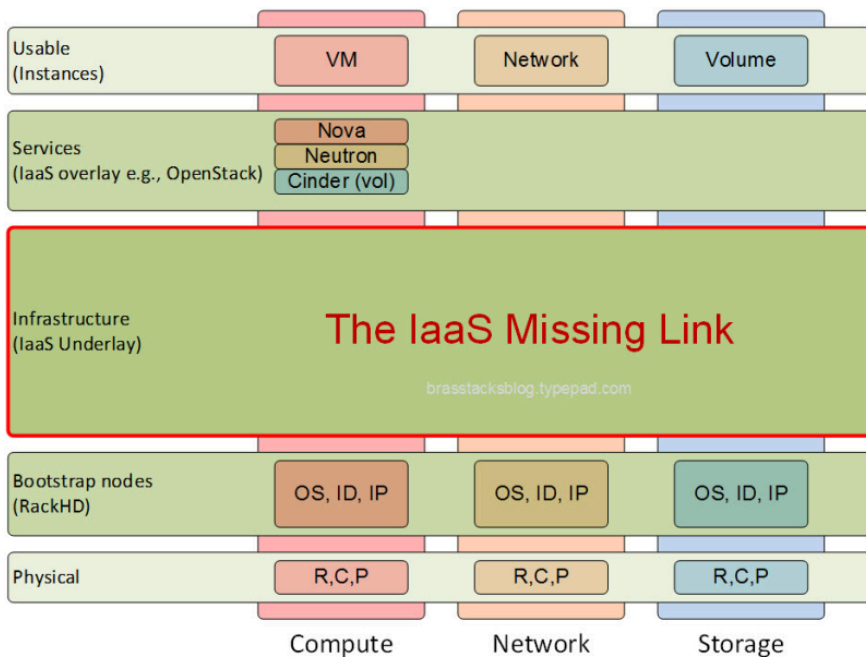
SEARCH

[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.



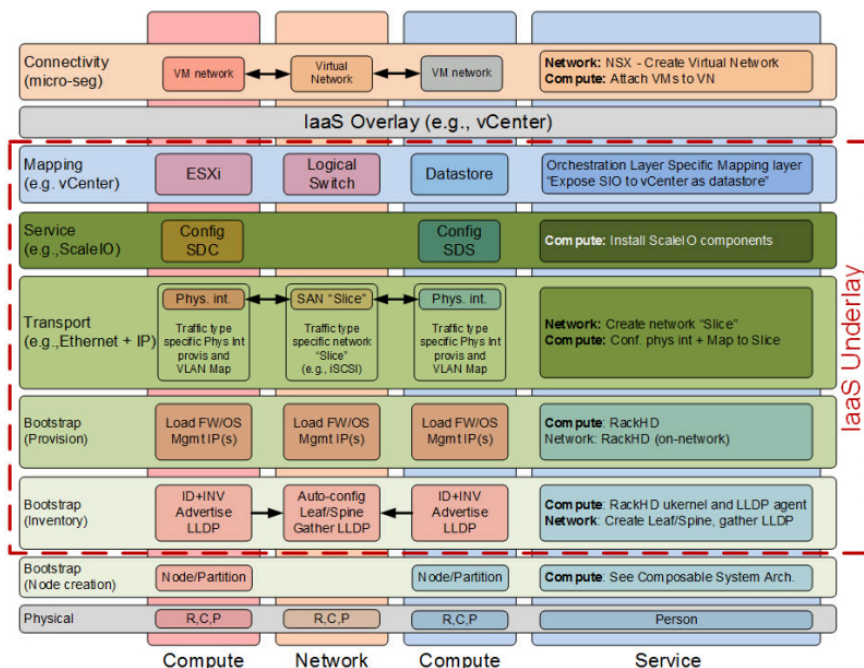
The IaaS Overlay

Most of you are probably very familiar with the IaaS Overlay and the IaaS Overlay Management and Orchestration (M&O) software used to control it. A couple of examples would be VMware vRealize Automation, OpenStack and whatever Amazon is using to orchestrate Amazon Web Services (AWS) more specifically EC2.

Based on the work we've done and the research we've seen performed by others in the industry, I believe the IaaS Overlay is all about the well-known axiom "Abstract, Pool, Automate". Judging by the solutions I see available for use in the Enterprise, I think many others would say the same.

The IaaS Underlay

Most of you are probably not as familiar with the IaaS Underlay and the IaaS Underlay M&O software used to control it. I would LOVE to provide examples, but we've been working in this space because we haven't found a solution (suitable for on-premises use in the Enterprise) that does everything we need. And by everything, I mean everything in the red (dashed) box shown below.



The IaaS Underlay explained

The diagram above can be broken down into:

- Columns that represent resources (e.g., Compute and Network) or actors (e.g., people or services that perform a particular function); and
- Rows that represent logical layers of the infrastructure as well as the configuration steps that are common to the resources in each column.

An aside: You might ask "what happened to the "Storage" column you were showing in [the previous blog post](#)?" and that's a phenomenally interesting story that will have to wait for my "post-retirement" book. That said, the removal of the storage column is one reason for the name change to ZTIP. The other primary reason is; we've been focusing on Hyper-converged solutions for full stack automation. This is because the concept of automation is something that traditional enterprises still seem to be evaluating, whereas the HCI community seems to have embraced it fully.

For the remainder of this blog series, I'll explain each of the layers (rows) in the above diagram and I'll start from the bottom and work my way to the top.

Before I continue, I'd like to share an observation that was made during the course of our work. Essentially, we noticed that the lower we went in the stack, the harder it was to automate. I'll provide more detail about this when I get up to the mapping layer case study, but I think this is a big reason why so few people have attempted to fully automate the IaaS underlay.

The Physical Layer

Although everything ultimately runs on physical resources, I don't consider the physical configuration of the components to be within the domain of the IaaS Underlay M&O controller. That said, we should at least mention that fact that before any of these components can be configured, each of them will need to be Racked, Cabled and Powered (R,C,P). This is a process that will be performed by a person, at least until the singularity, and at that point Robots

will be people too (and even they will probably be asking “isn’t there some way we can automate this?”).

Bootstrap - Node Creation

Once the nodes have been Racked, Cabled and Powered, a body of work comes into play that I’ll refer to as Composable Systems. The basic idea is that you will eventually be able to dynamically select CPU, Memory, Storage, GPUs, etc from pools of resources and then instantiate a “virtual” bare metal server that has exactly the right requirements for your application. It’s an area that is still in its infancy but [this blog post by Dell’s Bill Dawkins](#) contains some great additional information.

Because this area is still so new, I don’t currently include it when I talk about the IaaS underlay. That said, once a “Server Builder” API is available, it would make sense to include it.

Bootstrap - Inventory

Today, the lowest layer of the IaaS Underlay is the Bootstrap Inventory layer and the first bit of configuration that will need to be done in this layer is to configure the network.

Network Configuration (Auto-config Leaf/Spine + gather LLDP)

As will become clear as we move up the IaaS stack, there are all kinds of causality dilemmas (chicken or the egg scenarios) when trying to bootstrap Infrastructure and many of them can be solved by understanding how the elements you are trying to configure are related to one another, or put another way, how they are interconnected. I refer to these interconnectivity details as “topology information” and to properly understand the topology, I believe it makes sense to use the network as the source of truth. (h/t to Nick Ciarleglio from Arista networks for this insight)

However, before we can understand the topology, we first need to configure the network elements that will be providing the connectivity and hence we have our first causality dilemma (e.g., how do we configure the network if we don’t know exactly what it will be used for?) One approach that can be used is to configure the network in stages, and the first phase is something that I’ve been referring to as “IP Fabric formation”.

IP Fabric formation is basically just a way to say we are going to configure the switches so that they have basic connectivity between themselves.

With regards to the IP Fabric formation process itself, there seem to be three primary ways to accomplish this task:

1. Acknowledge it’s a huge pain-in-the-rear to automate basic network connectivity and just configure the network manually!
2. Use a discovery protocol to determine the physical connectivity information (i.e., how you physically connected the switches together) and then use this information to determine how to form the fabric.
3. Buy a solution that does this for you! Three solutions that provide this functionality, and that I also have at least a passing familiarity with, are: Big Switch, Plexxi and Mellanox (Neo).

We’ve done it all three ways:

1. Manual or template configuration is a good approach to use if you have a standardized network topology.
2. Using topology information to [determine how to configure the network](#) is possible if you know something about your network's characteristics and you have some dev resources to spare. For a bit more information about how you might accomplish this see the Network Configuration Example (below).
3. I think buying a solution rather than attempting to DIY a fabric controller makes the most sense for the vast majority of use cases. Of the Implementations that I mentioned above (i.e., Big Switch, Plexxi and Mellanox), I have the most experience with Plexxi and I PERSONALLY really like their BW limit feature. *Disclosure: We've written a [white paper](#) with Plexxi on the topic of Secure iSCSI SANs and used BW limits in that paper.*

End device discovery (ID+INV Advertise LLDP)

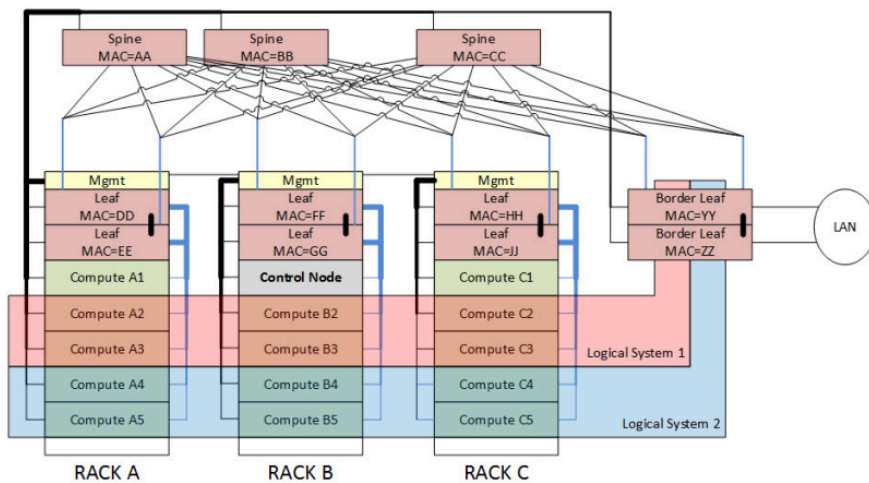
Once the switches have a basic configuration on them and basic connectivity established between them, you can do a couple of very interesting things:

1. Gather information about the connected end devices. We used LLDP in our [ZTSP PoC](#) and it seems like a really good approach to use if the end devices that you'll be attaching to your network actually support it.
2. Once you determine where the end devices are attached (LLDP), you can ID and inventory (INV) them. We use [RackHD](#) in all of our PoC's and it does everything we need it to. One slight caveat to keep in mind is that it currently performs the inventory by downloading a uKernel via PXE and this also requires the use of DHCP. As a result, you'll need to either allow for this traffic over a "default" VLAN or do some routing tricks if you're using an L3 Leaf/Spine. That said, we've also experimented with using Dell's iDRAC interface for the purposes of performing inventory over the management network and this seems like a really interesting approach. I should also point out there are other approaches (e.g., [Razor](#)) that can be used in place of RackHD.

So with the above in mind, let's look at an example that describes (at a high level) some of the work we've done in this space.

Network Configuration Example

The following configuration of Compute and Network resources will be used throughout this blog post series.



This configuration consists of:

- Three **Spine switches**. (i.e., MAC Addresses of AA, BB and CC)
- Three pairs of **Leaf Switches** that are intended to be MLAG'd together. (i.e., MAC Addresses of DD/EE, FF/GG and HH/JJ)
- A pair of **Border Leaf switches** that are intended to be MLAG'd together. (i.e., MAC Addresses of YY/ZZ)
- Some number of Connections to the customer's LAN
- An L2 **Management Network**
- Fourteen General Purpose **Compute nodes**.
- One **Control Node** where we will assume the **Centralized Network Control Point** will be running.

Overall Assumptions

- The Network hardware elements to be configured (leaf and spine switches) can be managed from a "centralized network controller". In this case, we'll assume that we're going to "roll our own" versus buying one.
- The Control Node will:
 - be physically connected to the management and Leaf switches that physically reside in the same cabinet as the Control Node itself.
 - provide a DHCP service
- Power has been supplied to all of the cabinets and switch hardware including the spine switches that are not shown as residing in a cabinet in the example configuration diagram.
- The Control Node will power up and the Centralized Network Controller will be able to discover (e.g., via LLDP) at least the management and leaf switches that it is directly connected to.
- The switches have obtained an IP Address from the DHCP server running on the Control Node

Phase 1: IP Fabric formation

Phase 1 assumptions

1. All of the switches are at their factory default settings,

2. The user has obtained at least one MAC Address of one of the Spine switches (e.g., by examining a label on one of the switches). This information is optional and will only be needed if the end user wants to verify the correct roles have been assigned to each switch type.
3. The switches have a NOS preinstalled on them (but this may not always be the case).

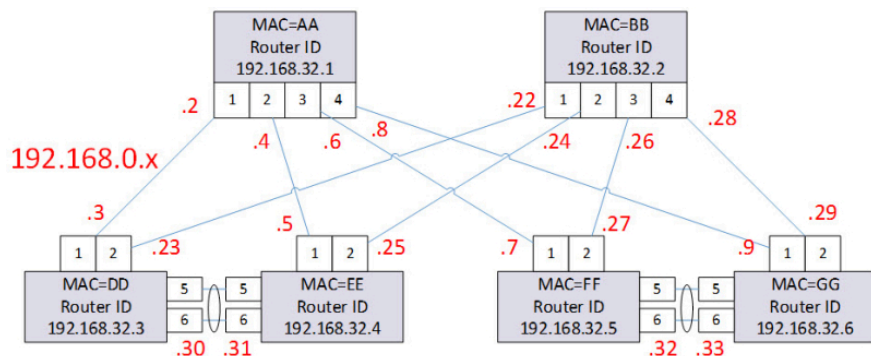
Topology discovery

1. When the switches first boot, they should attempt to download a configuration from the centralized network controller (e.g., POAP, ZTP, ONIE+). If a switch's role is unknown (e.g., unknown switch MAC Address), the Network Controller will associate the "Topology Discovery Configuration" with it and the switch will use this configuration until after Topology discovery has been completed and a role has been assigned to each switch (e.g., Spine, Leaf, Border Leaf). NOTE: The switch role (e.g., Spine, Leaf, Border Leaf) could also be set at the factory.
2. Once every unknown switch is running the Topology Discovery Configuration, the LLDP information being received by each switch can be stored in a centralized topology database.
 - a. Note, we used MongoDB for this purpose in the ZTIP PoC.
3. The network controller can use the process defined in [this blog post](#) to attempt to determine the role of each switch that has been discovered. The roles that have been assigned to each switch can be modified later during the IP Fabric configuration process.

IP Fabric configuration

1. The user launches the Fabric configuration wizard. See the [ZTIP Demo](#) at timestamp 1:38 for more information.
2. When the Fabric configuration wizard is launched, the user will be allowed to:
 - a. Set / modify the switch roles (e.g., leaf, spine) in this configuration. The wizard could obtain a list of switches and their discovered or factory preassigned roles from the centralized topology database.
 - b. Use or override the pre-provided pool of IP Addresses that will be used for IPv4 fabric (switch to switch) links.
 - c. Use or override the pre-provided pool of IP Addresses that will be used for the router IDs.
 - d. Use or override the pre-provided pool of IP Addresses that will be used for the creation of VLAN interfaces on the switches in the environment.
3. Once the switch roles have been determined, the IP Fabric configuration service will use the switch role and topology information stored in the centralized topology database to create a candidate IP Fabric topology like the one shown below. Please note, the following diagram is just a simplified version of the example configuration shown above. Also note that IPv4 Addresses have been assigned to each switch interface and a router ID has been provided to each switch as well. We could have used IPv6 but have

encountered a switch vendor specific issue that prevented us from doing so during testing. In any case, this is merely an example of what could be done, not what should be done.



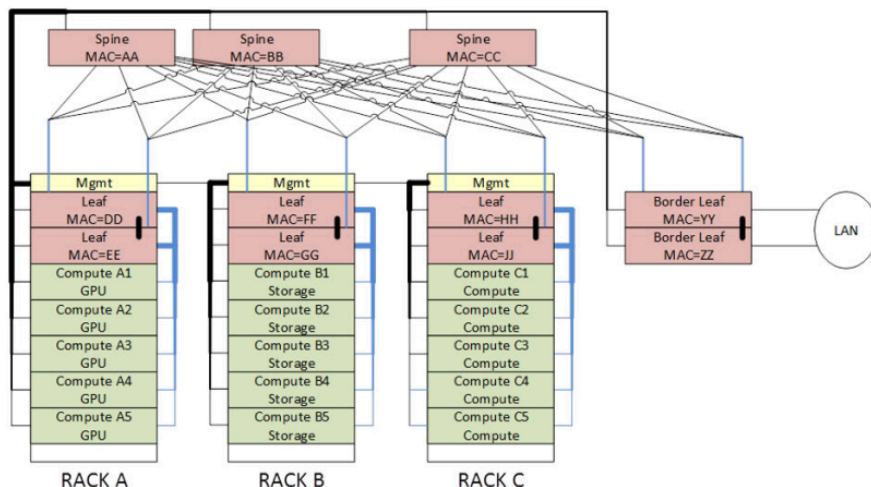
- Once the candidate IP Fabric topology has been created, the network controller can create configurations for each of the switches in the topology.

- Configure Leaf to end device links

Note: Initially all end device interfaces (e.g., eth7 and above) could be put into a default VLAN (e.g., 4001) for the purposes of PXE boot and inventory. This will allow the hosts to obtain an IP Address, PXE boot and then perform inventory.

- Repeat the above steps for each switch that was discovered. When finished each switch should have a configuration file associated with it.
- The switches could now be restarted and download their configuration files from the centralized network controller or something like Ansible or Puppet could be used to set the switch configuration.

At this point the IP Fabric has been formed and the compute resources should be able to PXE boot and download the RackHD microkernel to start the inventory process. We will assume that once the inventory process has completed, the capabilities of each node are as shown below.



Note that each rack contains homogeneous node types but this won't typically be the case. Also note that "GPU" indicates that the node contains GPUs, while "Storage" and "Compute" indicate that the nodes are either Storage "heavy" or Compute "heavy".

The remainder of the network configuration process (e.g, Slice creation) will be handled as workloads are on-boarded and will be discussed in the next blog post.

Thanks for reading!

Posted at 11:04 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](#)