

# Politecnico di Torino

Master Degree course in Communications and Computer  
Networks Engineering

Master Degree Thesis

## Engineering Data Centers



### **Supervisor**

Prof. Fulvio Risso

### **Candidate**

Peiman Roufarshbaf

Academic Year 2020



## Contents

<b>Chapter 1</b> .....	1
<b>Introduction</b> .....	1
<b>Chapter 2</b> .....	4
<b>Global design and architecture</b> .....	4
<b>2.1 Current network conception problems</b> .....	4
<b>2.2 SDN (Software Defined Networking)</b> .....	6
<b>2.2.1 Application layer</b> .....	6
<b>2.2.2 Control layer</b> .....	6
<b>2.2.3 Infrastructure layer</b> .....	8
<b>2.3.1 Tools and components</b> .....	10
<b>2.3.2 VMware Platform Service Controller</b> .....	11
<b>2.4 Global Architecture</b> .....	11
<b>2.4.1 Distribute Resource Scheduler</b> .....	13
<b>2.5 High Availability (HA)</b> .....	17
<b>2.6 Fault tolerance</b> .....	20
<b>2.6.1 Fault tolerance Requirements</b> .....	20
<b>Conclusion</b> .....	23
<b>Chapter 3</b> .....	24
<b>Network Virtualization with NSX-V</b> .....	24
<b>3.1.1 Standard Switch (vSwitch)</b> .....	24
<b>3.1.2 Port Group</b> .....	25
<b>3.1.3 Distributed Switch</b> .....	26
<b>3.1.4 VXLAN</b> .....	27
<b>3.1.5 Transport Zone</b> .....	27
<b>3.2 Virtual network architecture</b> .....	28
<b>3.3 Distributed Logical Router (DLR)</b> .....	30
<b>3.3.1 Overview</b> .....	30
<b>3.3.2 DLR Components</b> .....	31
<b>3.4 Deploying DLR to architecture</b> .....	33
<b>3.5 Deploying Dynamic routing algorithm on DLR</b> .....	35
<b>3.5.1 NSX Edge Gateway</b> .....	35
<b>3.5.2 Edge Service Gateway</b> .....	35
<b>3.5.3 Deploying BGP protocol</b> .....	36

<b>3.6 Distributed Firewall and micro-segmentation</b> .....	39
<b>3.6.1 Service composer</b> .....	39
<b>Conclusion</b> .....	42
<b>Chapter 4</b> .....	43
<b>NFV and NSX-T</b> .....	43
<b>4.1 NSX-T components</b> .....	43
<b>4.2 The virtual Network architecture in multi-hypervisor</b> .....	45
<b>4.3 NAT and DHCP with NSX-T</b> .....	49
<b>4.4 Dynamic host configuration protocol (DHCP protocol in NSX-T)</b> .....	51
<b>4.5 Load balancing in NSX-T</b> .....	53
<b>Conclusion</b> .....	55
<b>Chapter 5</b> .....	56
<b>vRealize Network Insight</b> .....	56
<b>5.1 Traffic distribution</b> .....	56
<b>5.2 VMware Network Insight information collection mechanism</b> .....	57
<b>5.3 Application discovery</b> .....	62
<b>Conclusion</b> .....	64
<b>Bibliography</b> .....	65



## **Acknowledgments**

Firstly, I would like to express my thanks to my supportive supervisor, Mr. Fulvio Risso, who has supported me throughout this thesis. Then, I must express my very profound gratitude to my family for supporting me in all the steps of my life with their continuous encouragement. These accomplishments could not be possible without them. Thank you.



# Chapter 1

## Introduction

In recent years the demand for internet services has experienced a severe increase; to satisfy this demand, we need physical facilities known as data centers. With the introduction and development of Software-Defined Networks (SDN) and Network Function Virtualization (NFV), the world of networks experienced undergone changes. Meanwhile, SDN, by separating the control plane and data plane, not only made it possible to have more comfortable and centralize management over the infrastructure but also provided scalability and flexibility, which are very important in designing data centers. VMware is the main leader in virtualization technology, and by producing vSphere, vCenter, and NSX-T/NSX-V plays a critical role in leading our networks to SDN. NSX-V is a software-defined networking solution specific to vSphere hypervisor that allows network administrators to dynamically initialize, manage, control, and deploy different network policies. NSX-T provides all these functions and features plus supporting multi-hypervisors. Network Function Virtualization (NFV) is related to deployment and managed network services, which are entirely implementing in software. For example, load balancer, Network Address Translation (NAT), Firewalls.

In this work, by using VMware products, we are going to Engineering and designing a Data Center. In the first chapter, after reviewing the current network problems and describing how virtualization solved these problems, we will use vSphere 6.7 and vCenter to reach a central management point. After that, we will use this centralized management to deploy services such as; DRS (Distributed Resource Scheduler), High Availability (HA), Fault Tolerance (FT), and vMotion to have a robust and flexible network.

The second chapter is dedicated entirely to Network Virtualization. To have a centralized network and security management platform and creating different policies to spread among underlay network components, we deploy NSX to vCenter. NSX gives us the flexibility to create a uniform distributed switch among all hosts and provide East-West connectivity. After that, by creating different VMkernel port groups, we will separate network and management traffic. At the end of the chapter, we will apply DLR to our network design for providing connectivity between different subnets. Also, we will configure edge routers with dynamic routing protocol (**Border Gateway Protocol**) to provide north-south connectivity to outside networks. In the end, by using distributed firewalls, we provide different policies for different segments.

In chapter three, we will add a KVM host, and then by using the VMware network virtualization tool for multi-hypervisors (NSX-T), we will apply different Network Function Virtualization (NFV) for various purposes. For example, load balancer to distribute client requests across multiple servers, Firewalls to control and monitor incoming -outcoming traffics, and Network Address Translations (NAT) to translate private IP addresses to public IP addresses and vice versa.

Chapter four is dedicated to vRealize network insight. It is a monitoring product of VMware that brings intelligent operations for SDN and enables global view across both virtual and physical networks and gives administrators the ability to find pinpoint of the problems. In this chapter, we are going to see how we can use this product to monitor our infrastructure and solve network problems.



## Chapter 2

# Global design and architecture

### Overview

In this chapter after reviewing problems of traditional networks and describing how software-defined networking solved these problems we are going through the virtualization and by using VMware vSphere we will be able to virtualize our hosts and virtual machines, after that, we will deploy vCenter to provide centralized management to the overlay network. In this chapter, our global design contains two Datacenters with several hosts, and virtual machines, each of these data centers contains their shared storage which is shared internally between their hosts, and each host can access this shared storage. Then we are going to use our virtualized infrastructure to do operations that we were unable to do in traditional networking, for example; Distribute resource scheduler (DRS), High Availability (HA), Fault Tolerance (FT), and vMotion

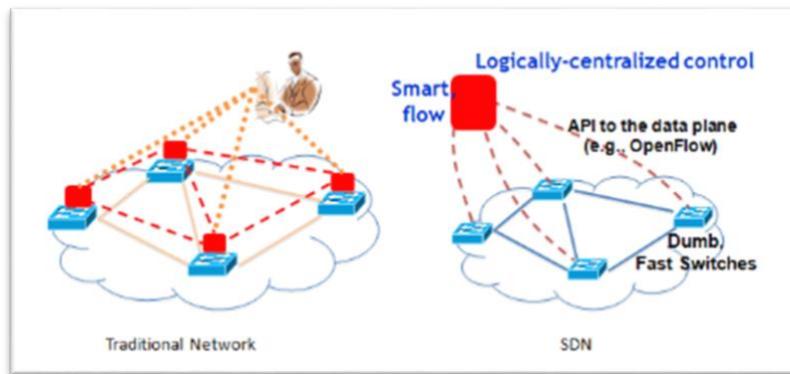
## 2.1 Current network conception problems

Before proceeding to know about SDN, it needs to understand the problems of traditional networks and why this technology discovered. In traditional network architecture, there is no separation between control-plane and data-plane; also, the burden of decision-making based on our entire network falls on each networking device independently. Each device has a role, and each of them should independently decide where they need to send traffic. Therefore, none of these devices used to know the whole map of topology, just a little part and its end to end path.

Another problem with decentralized topology was a high degree of complexity, especially in administration, troubleshooting, and scalability. Traditional networks usually have a static architecture with high level of complexity that made it hard for administrators to configure each device of different vendors for different roles, and the risk of misconfiguration or causing unwanted behavior in our system always exists. Not only is configuration difficult and time-consuming, but also troubleshooting needed to be done one by

One on each device. After all, when we reached our goal throughput, we needed to configure all devices again to have slight changes in configurations.

Resource allocation was another challenge in traditional networking, especially when the company was growing fast. On the one hand, if we assigned over-allocated resources, we had wasting resources; on the other hand, under-allocating obliges us to rebuy hardware. SDN is the solution for all these problems, and it solved traditional network problems by separating the control-plane and data-plane. In paradigm 1.1 depicted the difference between SDN and traditional networking.



paradigm 1.1 \_ Difference between SDN and traditional network architecture

## 2.2 SDN (Software Defined Networking)

Software-Defined Networking (SDN) solves traditional networks' problems and brings us dynamic, programmatic efficient network configuration for network performance and monitoring, thanks to the centralization of control logic. This separation has several advantages:

Different applications can integrate into each other. For example, load balancing and routing applications can be used simultaneously with different priorities in the edge routers.

Applications can have a global view, and control-plane software modules can make more efficient policy decisions based on this information.

It gives us an affordable and non-complex testbed to be used for deploying SDN applications. The possibility of sharing or moving the abstractions of the network programming language and the platform of the control-plane will make it easier to program the related applications.

From the functional point of view, SDN can divide into three sub-layers, which illustrate in Figure 1.2.

### 2.2.1 Application layer

The application layer interacts with the control plane, and it is responsible for building an abstract view of the network by collecting controllers' information via Application programming interfaces (API) for decision-making purposes. These applications could be management or business.

### 2.2.2 Control layer

The control layer is responsible for configuring the infrastructure layer, and it does that by receiving service requests from the upper layer, which is the application-layer. The controller maps the service requests dynamically to the infrastructure layer in an optimal possible manner.

The controller is called the brain in our architecture, and this intelligence centralization has its drawback. For example, security, scalability, and dependability are topics that we should concern in this architecture. To protect this brain from failure, we usually have a cluster of controllers. VMware's suggestion is to deploy only a three-node controllers cluster.

## Northbound/southbound interfaces

Northbound interfaces are restful APIs used to communicate to a higher layer, which is the application layer, and southbound APIs are interfaces that communicate with switches, routers, and other components in the infrastructure layer.

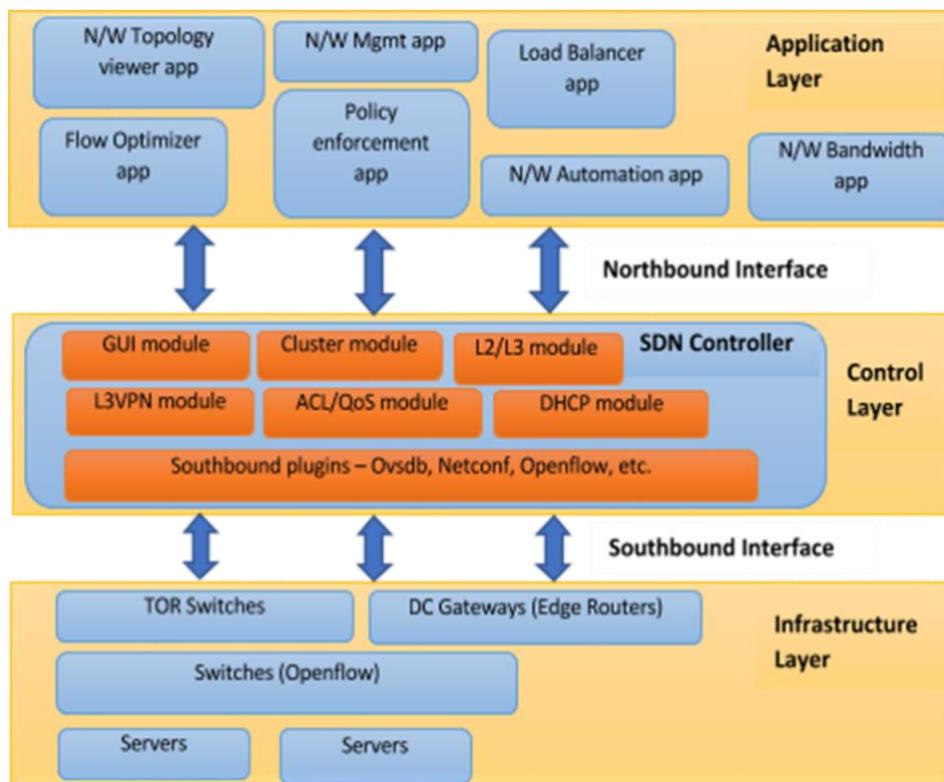


Figure 1.2 \_Simplified view of SDN architecture

### 2.2.3 Infrastructure layer

It composed of different equipment that forms an underlying network and responsible for forwarding network traffics. One of the most critical differences between infrastructure equipment such as routers, switches, Edge routers, and controller nodes are traffic passing.

Traffic is not passing through controller layer and in case of failure one of the controllers it will not affect our network connectivity, but if one of the components inside the infrastructure layer such as a switch, or router loses its connectivity it will affect our network, and traffic cannot pass through it.

## 2.3 VMware virtualization products

### Background and motivation

Virtualization is a technology that is increasing its demand because of its number of benefits, in the beginning, only one system used for each application for isolation purpose, and each system has a dedicated hardware resource, but this was not only an expensive solution for isolation but also was waste of system resources. These problems became a reason for the development of virtualization.

Virtualization gives us the ability to run multiple operating systems or applications on the same physical hosts and provide each of these hosts an abstract view of hardware resources. Virtualization gives us the ability to flexibly and dynamically allocate resources to each virtual machine, and this flexibility is beneficial for data centers that running internet applications, and their customers may see highly and unpredictable load. Before virtualization, we had to dedicate fix amount of resources such as RAM, CPU, and storage, and over-allocating caused a waste of resources and, under allocation, forced us to repurchase these resources. With virtualization, administrators can increase or decrease hardware resources very easily and fast. Therefore, virtualization optimized the resource usage and optimized space usage by compacting several servers into one physical server, and this reduction of servers leads to many benefits, especially when we scale up. In summary, the benefits of virtualization are:

- Server Consolidation
- Improve resource Utilization
- Scalability and reliability
- Business continuity

Virtualization added one extra layer on top of the hardware layer in traditional network architecture, which is called "hypervisor." Hypervisor makes us able to install and run multiple guests with different operating systems running in parallel on a single server with limited hardware resources and associate to each of these virtual servers a specific hardware resource such as; RAM, CPU, Storage, Etc. (Yaqub, March 23, 2012)

Hypervisor gives the illusion that each VM owns its resource, and it is not shared. This compact approach leads our topology to cost and space reduction in one hand and, on the other hand, optimize utilization, which is shown in [Figure 1.3].

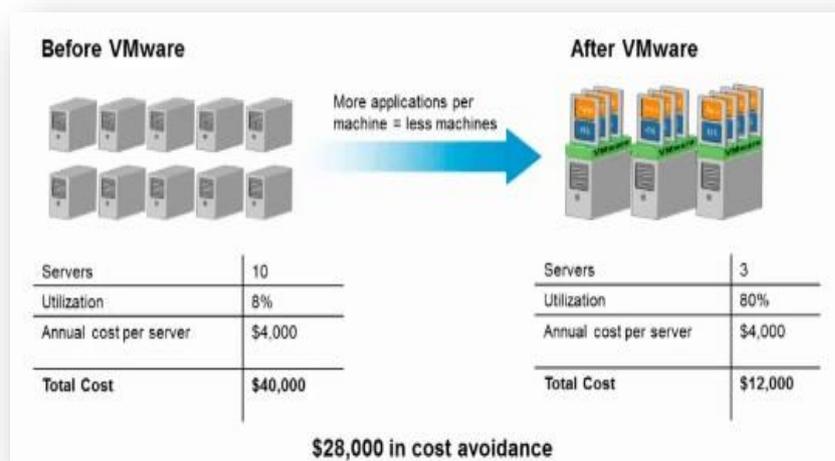


Figure 1.3\_ Comparing costs and utilization before or after virtualization

There are two types of hypervisors. The first type, which is known as a bare metal or hypervisor type one, installing directly on the host's physical hardware. The second one, known as hypervisor type two, is an application installing on an existing operating system.

Examples are vSphere ESXi, Microsoft Hyper-V, and Linux KVM for type one, and VMware Workstation, parallel for type two.

## 2.3.1 Tools and components

### vSphere and vCenter

VMware provides different tools and GUI for various purposes. VMware vSphere is an umbrella term platform, including VMware bare-metal hypervisor ESXi, vSphere Web/Flash Client, and vCenter.

vSphere and vCenter are components of VMware vSphere suite; meanwhile, ESXi is a type-one hypervisor which plays a fundamental role in virtualization by providing an environment for installing and running multiple applications with different operating systems in parallel. To install and manage each of these hypervisors, we need to connect via vSphere Web/Flash client.

vCenter is another part of the vSphere suite, which gives us the ability to manage and configure all these hosts and virtual machines by providing a centralized console for operation, resource provisioning, and management.

There are two ways to manage all of the virtualized hosts; one way is to use vSphere web/flash-based and individually configure each host, which is time-consuming and not practical in data centers or install a vCenter and join all hosts to the vCenter and configure them centrally as it depicted in Figure 1.4. As we need to manage many servers and applications, we choose the second option.

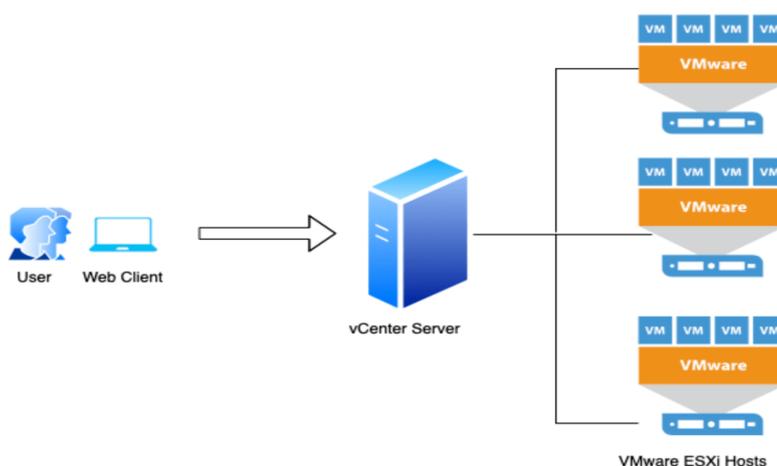


Figure 1.4

There are two methods to deploy vCenter. The first method is to install a vCenter appliance, which is a preconfigured Linux based virtual machine or installing vCenter on a windows server. We almost always use the first option.

## 2.3.2 VMware Platform Service Controller

PSC is a new service introduced after the vSphere 6 version, and it is responsible for handling security services such as; licensing, certificate management, server reservation, and single sign-on. PSC is a distributed service with a maximum number of eight instances for high availability, which achieved through load balancing technology and providing replicates information such as permissions and roles to other PSC instances. It can be installed internally or externally within the installation of vCenter.

Single sign-on is an appliance and authentication broker in PSC with a security token exchange infrastructure. When a user authenticates via vCenter single sign-on and receives a SAML token, then by using this token and receiving a privilege, the user will not need any additional authentication to login on different applications.

Single sign-on is the only component that can be installed beside PSC. These following components integrate with PSC:

- VMware Syslog health service
- VMware Directory service
- VMware Authentication framework
- VMware Service control agent
- VMware Common logging service
- VMware HTTP reverse proxy
- VMware identity management service
- VMware certificate service

## 2.4 Global Architecture

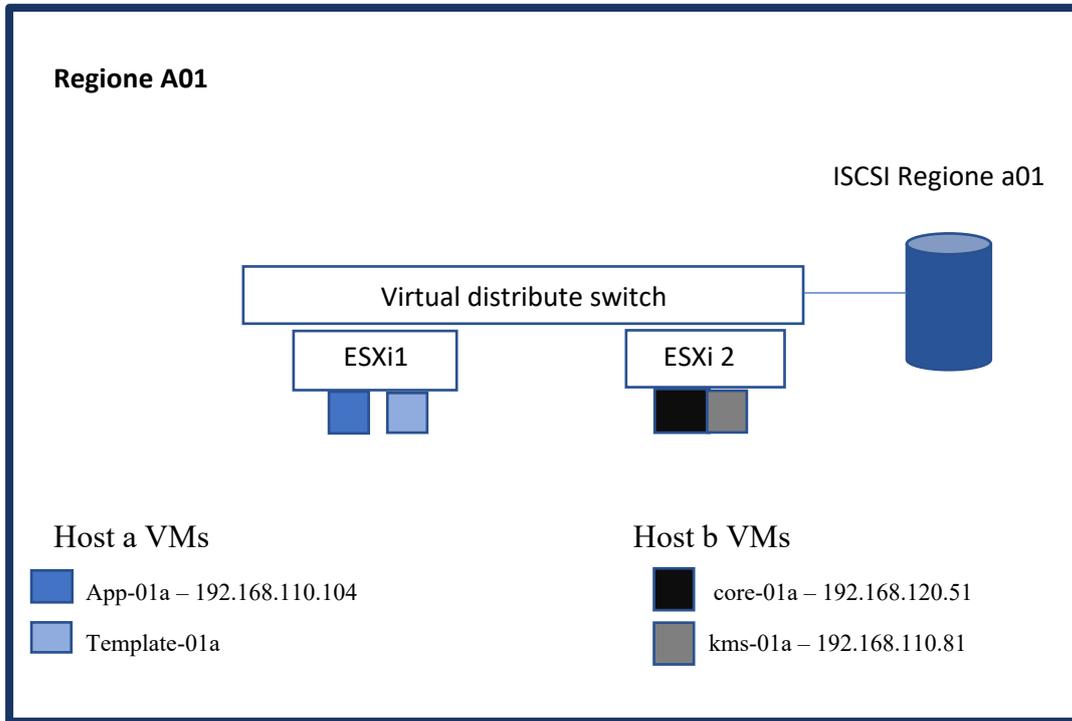
In our global design, we have two Data centers, and each of them has a dedicated ISCSI shared storage. Inside of each data center, there is a distributed virtual switch that provides connections between hosts [Figure 1.5].

In each Data Center, we have several clusters. The reason behind of clusters separation is when we dedicate several hosts to a cluster; the resource becomes a pool

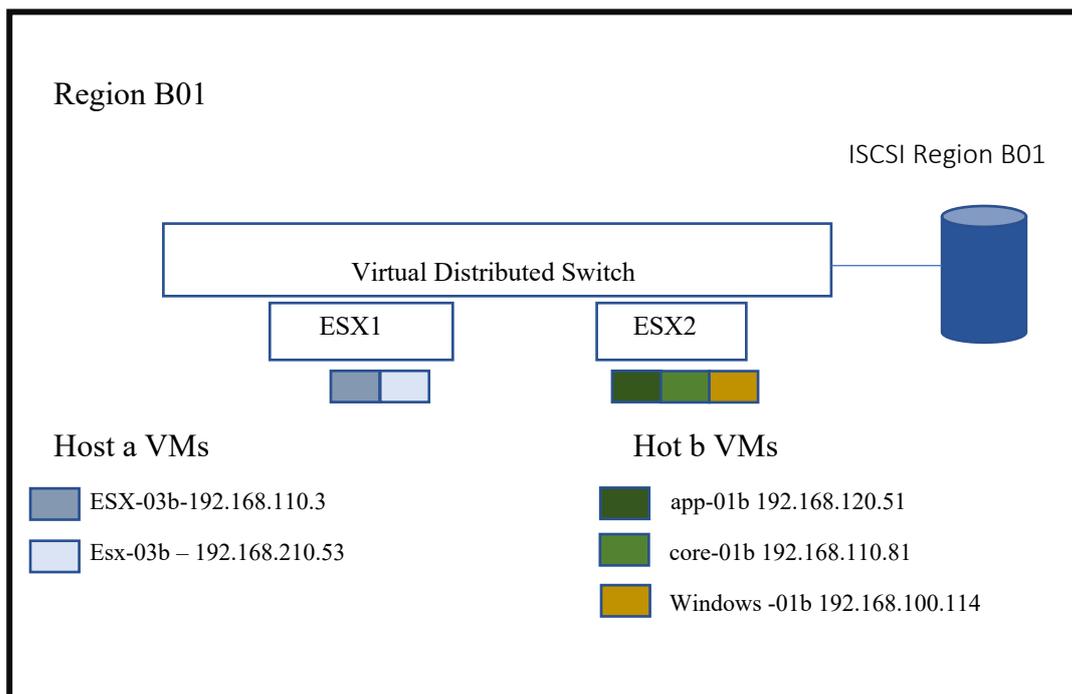
of resources. It gives us the flexibility to use features like DRS, HA, FT, and vMotion.

The global architecture of the first Data Center (VCSA-0a1)

[Figure 1.5\_ Global architecture of Data Center a and b]



The global architecture of the second Data Center (VCSA-01b)



So far, we deployed our vCenters and installed each of these hosts and VMs. [Figure 1.6](#)

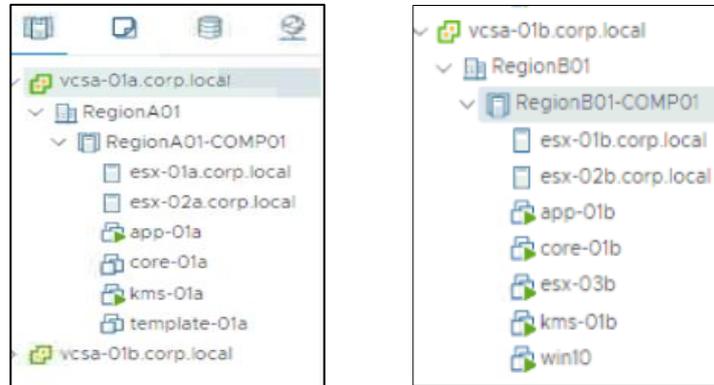


Figure 1.6

## 2.4.1 Distribute Resource Scheduler

After installing vCenters and deploying hosts a and b with their VMs, we are going to use DRS to automatically balance the memory and CPU loads between different hosts inside a cluster. We do this action by migrating VMs while they are running from an overloaded ESXi host to another host in the same cluster that has enough computational resources such as (RAM, CPU, Storage space).

To use this feature, we should make sure we have requirements:

- ✓ VCenter server needs to be installed.
- ✓ CPUs of ESXi hosts must be compatible (it is better to be the same vendor)
- ✓ Hosts should be part of the vMotion network in the cluster that we want to have DRS (creating and configuring of different port groups is covered in chapter two, for now, we consider hosts are participating in vMotion port group).
- ✓ All hosts of the cluster should use shared storage, which is accessible by all of them.
- ✓ Shared storage needs enough space to allocate all virtual disks and all the migrated VMs (Moenster).

At this step, our network has requirements we need to enable DRS, as we discussed earlier, DRS is a cluster base, and by going through the configuration tab of each cluster, we can activate and set the scheduler. With DRS, we have several scheduling options: fully automated, partial automate, or manually; we can also set the aggressive levels. We used a full automation scheduler with full aggression.

**Fully automated level:**

DRS works without human interaction and not only generates load balancing recommendations and initial placement, but also executes them automatically.

**Partially automated level:**

DRS will handle initial placement of virtual machines, but any further migration recommendations will be surfaced up to the administrator to decide whether to move the virtual machine.

**Manual automated level:**

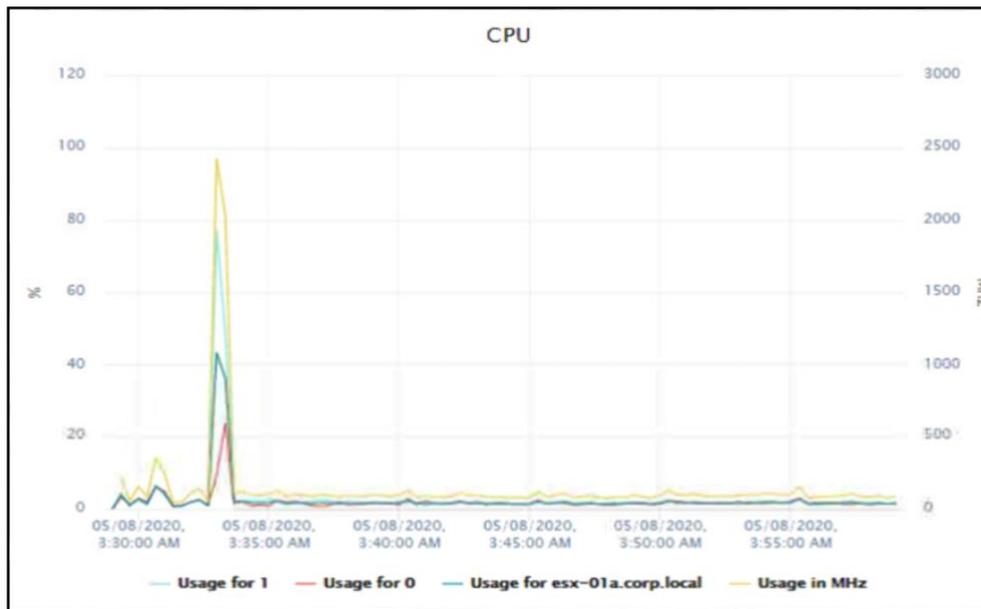
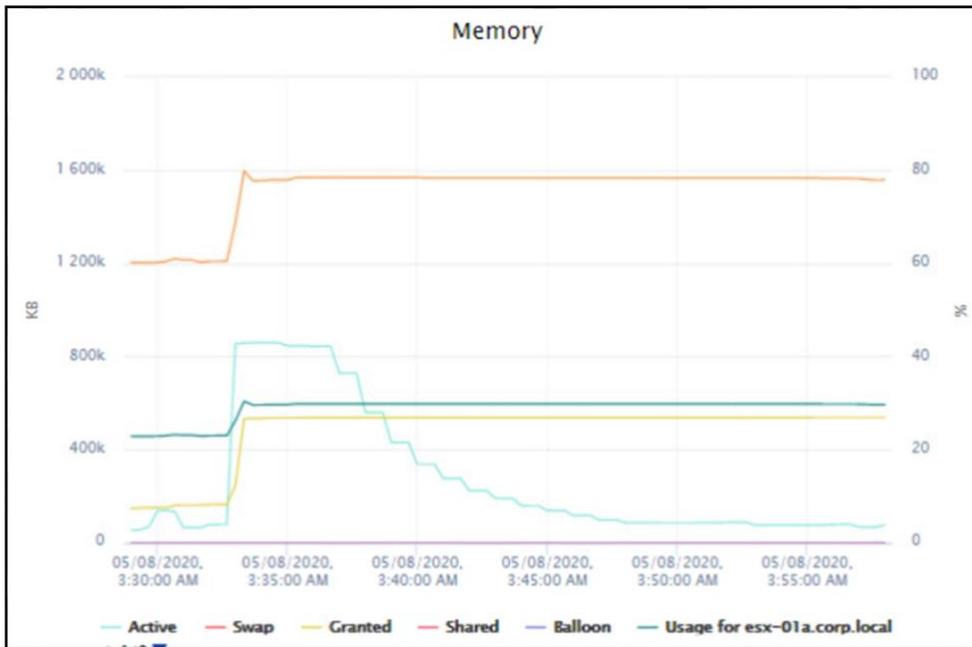
Both initial and load balancing are in manual mode, and the administrator should decide about them.

**Full aggressive:**

Full aggressive means whenever the load on the one host in the cluster increased DRS makes action and does vMotion immediately, but we do not use it in data centers because it generates lots of traffic and makes our systems unstable, also most of the time there is a temporary load, and we do not want to make our systems unstable.

DRS makes an action when there is overload, to verify and monitor this process when the loads on the system balanced between hosts, we used a piece of code on the command line of two virtual machines in host ESX01-a of data center\_01a to overload them and imbalance the cluster. To verify the action of DRS, we need to monitor performance before and after the overload.

In diagram 1.7, after setting up our configuration to fully automated with a high level of aggression, we monitored the real-time load of RAM and CPU of ESX-01a on 05/08/2020 between 3:30 AM - 3:55 AM to check our resource performance and usage before overload. As can be seen our memory and CPU usage are in stable mode [Figure 1.7](#)

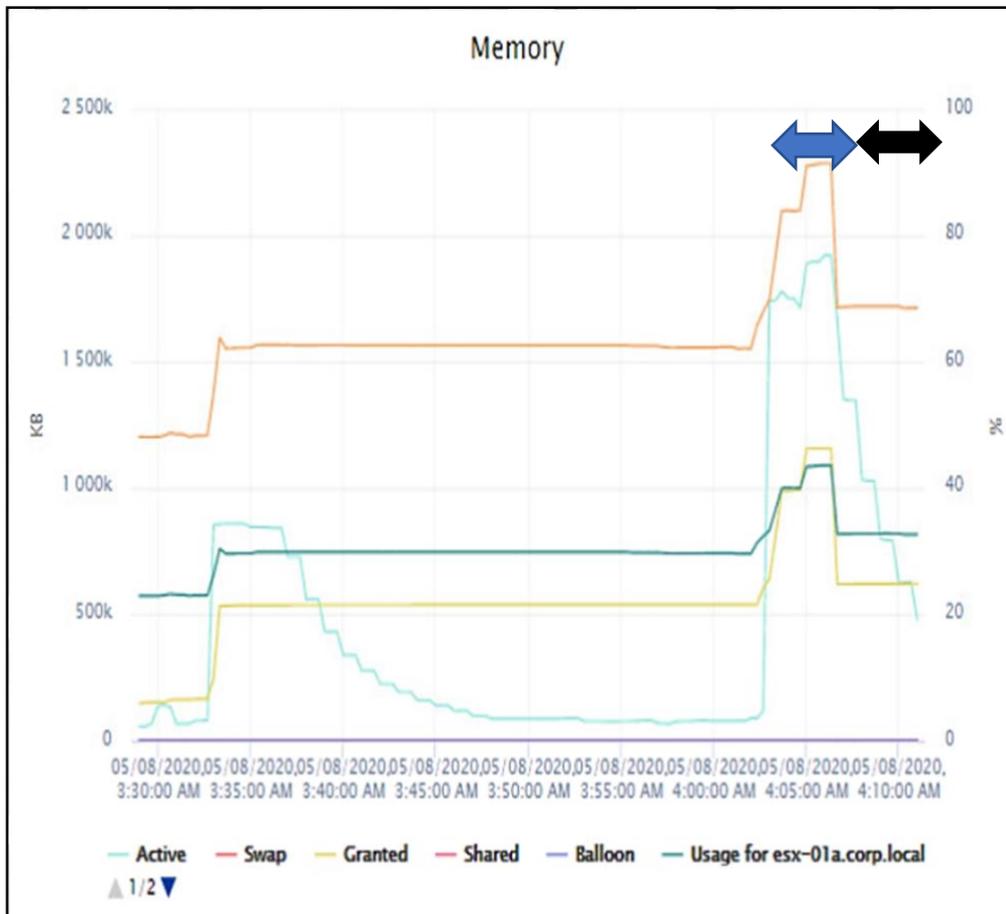


(Figure 1.7 - balance loads on ESX-01a)

Then we use code below in the command line of the application server and template server of host 01a to imbalance the host:

```
tc@core:~$ for i in 1 2 3 4; do while : ; do : ; done & done
```

After running this code, memory and CPU usage reaches the maximum level (between 80-90%) in blue intervals. Then, as we set our scheduler to aggressive mode, it forces the overloaded VMs to move to ESX-02a immediately at time 4:05 AM, after this migration we can see on black arrow that memory usage and CPU usage passed peak from high loaded hosts and here is a point that migration took action and these virtual machines transferred to host b.



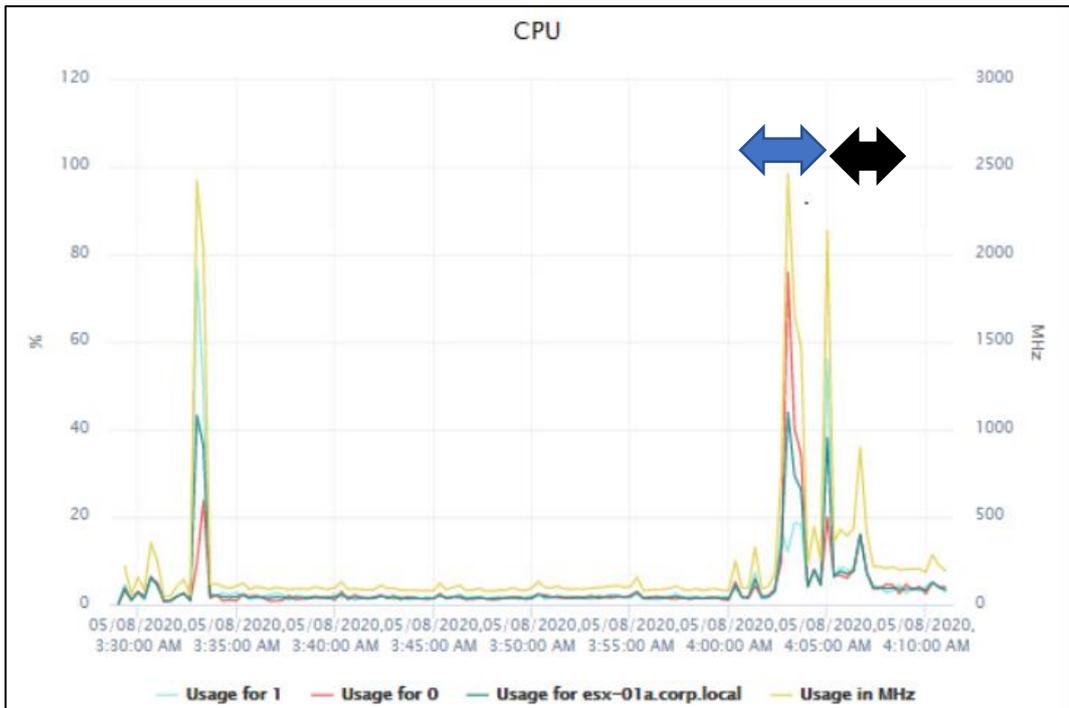


Figure 1.8

In the Syslog summary, we can verify that these two virtual machines migrated successfully at time 4:06:21 AM, and these migrations took seven and six (ms).

Figure 1.9

Migrate virtual machine	✓ Completed	app-01a	System	7 ms	05/08/2020, 4:06:21 AM
Migrate virtual machine	✓ Completed	app-02a	System	6 ms	05/08/2020, 4:06:21 AM

Figure 1.9

## 2.5 High Availability (HA)

VMware vSphere High Availability is another essential and useful utility in Data Centers, which is included in vSphere suite and delivers the ability required by applications running on virtual machines independent of their operating system and provides a uniform, cost-effective protection against applications and hardware failure. HA helps us to:

- Eliminate the need for dedicated software and hardware standby in a virtualization environment
- Decrease downtime
- Increase reliability and provide disaster recovery continuity

HA is another cluster-based utility beside DRS, which discussed in the previous section. When we create a vSphere HA cluster, one host elected as a master host, and

the rest of the hosts referred to as slaves. The master host communicates with vCenter and slave hosts to monitor the status of hosts and virtual machines in the cluster to protect all of them in case of any issues such as applications crash or host failures.

The master host can detect hosts and VMs failures through heartbeat mechanism by sending a signal every second to make sure they are running as expected. If the master host does not get this heartbeat, it determines if the host is heart beating to one of the cluster's datastore as a double-check, if the connection exists between the data store and slave, then the master node consider it as network isolation and failure of connection between master and slave. However, if the master node does not receive any heartbeat from slave and data-stores, it will consider as a slave failure.

Corrective action depends on the type of failure detected. If a VMs crashed, the host continues to run, and HA forces the VM's to restart in the originated host, but if an entire host fails the HA utility starts with highest priority continuing with lower priority until all to finished and restart all affected virtual machines on the other hosts in the cluster with all their configurations.

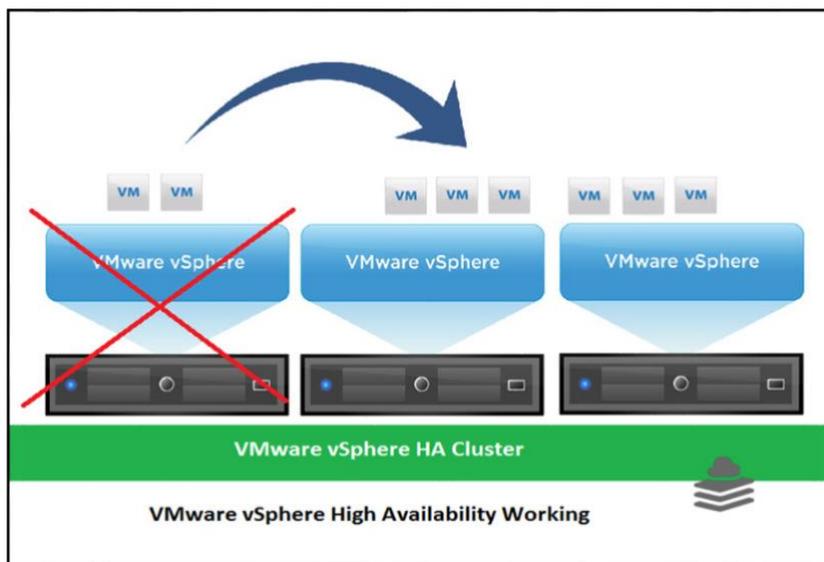


Figure 1.10 -HA

The reason after migration the virtual machines still can use the same IP address is each host connected to a dedicated port group in the distributed virtual switch, and by moving to another host, still, they are in the same port group of the same virtual switch.

Before configuring HA, we need some pre-checks:

- vSphere essential license
- Shared storage
- CPU compatibility
- Enabled DRS

We have shared storage as depicted in [Figure 1.5](#) and enabled DRS in the previous section, and all hosts have Intel CPU, so all HA requirements are satisfied.

We configured our High Availability to restart the application server in case of failure on the other host, to verify this configuration, we turned off the host to mimic host failure, and then we can see both application server and template-01a disconnected and cannot receive any signal.

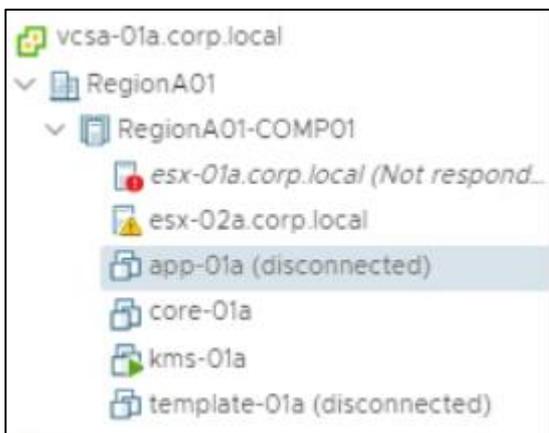


Figure 1.11

Then HA makes action and restarts disconnected servers on the other host. As we can see in [figure 1.12](#), although the host-01a is in maintenance mode, the application server (app-01a) is running on the other host.



Figure 1.12

## 2.6 Fault tolerance

Fault tolerance is another failover protection utility that we can deploy, the most significant advantage of FT in comparison to HA is 100% uptime, and end-users do not lose connectivity by providing primary and secondary VM. These two are sync, and secondary VM takes over in the event of a host failure.

Mostly we use FT for critical business-virtual machines like banking servers, although it has the best protection among failure protections, it has some disadvantages, which leads us to use HA in data centers.

- The VM must not have any snapshots
- Hosts must have licensed for vSphere FT
- FT-protection VM will use twice as much resource. For example, if the primary V uses 4GB of RAM, also the secondary VM will use 4GB

### 2.6.1 Fault tolerance Requirements

vSphere fault tolerance (FT) is made possible by four underlying technologies. These technologies are:

- Storage
- Network
- Transparent
- Runtime

#### Storage

FT ensures the primary and secondary storage of virtual machines are always sync. Whenever the FT protection begins, the initial synchronization of virtual machine disks (VMDKs) happens by using storage Motion to ensure both storages have the exact disk state. When the storage vMotion completes the FT, a virtual machine considered as FT-protected.

After synchronization, vSphere FT mirror VMDK modifications to keep the secondary identical.

## Transparent failover

To have a transparent failover, FT ensures that the state of the new primary is agreed with the previous one. That can be achieved by holding and only releasing externally visible output from the virtual machine once an acknowledgment made from the new primary affirming that the state of the two virtual machines is consistent.

## Runtime State

vSphere FT captures the active memory and exact execution state of the virtual machine to ensure that runtime of the two replicas are always identical. To decrease the transformation time of FT in case of failure, we need to dedicate a high-speed network to allow virtual machines instantaneously to switch from running on the primary ESXi to the secondary ESXi host.

## Network

To ensure that even after the FT, virtual machine identity, and all connections preserved, networks used by the virtual machines virtualized by the underlying ESXi host. Very similar to vMotion, FT manages the virtual mac-address as a part of the process. Since FT preserved the storage, exact execution state, and network identity with all active network connections, the result is zero downtime.

We configured the FT with one virtual machine in the second host as a secondary for an application server in the host (a), which is primary. Figure 1.13

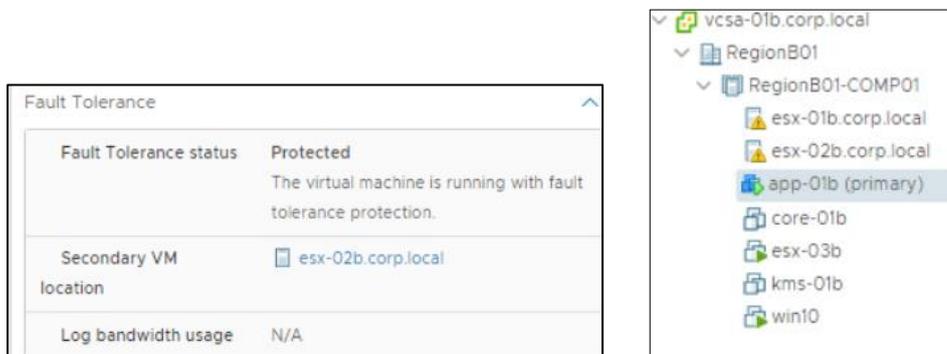


Figure 1.13 – The primary and secondary FT components

Now we need to test fault tolerance to be sure that is configured and running well; for this reason, there is an option "Fault Tolerance Test.". After running the FT test, we expect to see the secondary machine takes over, and the application server with all the network configuration and IP addresses runs on the other host. Figure 1.14

Start Fault Tolerance Secondary VM	✓ Completed	app-01b	System
Power Off virtual machine	✓ Completed	app-01b	
Test failover	✓ Completed	app-01b	
Start Fault Tolerance Secondary VM	✓ Completed	app-01b	

Figure 1.14 – FT Test

app-01b		app-01b	
Summary	Monitor	Configure	Permissions
 <p>Powered On</p>	<p>Guest OS: Other 3.x Linux (32-bit)</p> <p>Compatibility: ESXi 5.5 and later</p> <p>VMware Tools: Running, version: 2.0.1</p> <p>DNS Name: core</p> <p>IP Addresses: 192.168.220.104</p> <p>Host: esx-01b.corp.local</p>	<p>Guest OS: Other 3.x Linux (32-bit)</p> <p>Compatibility: ESXi 5.5 and later</p> <p>VMware Tools: Running, version: 2.0.1</p> <p>DNS Name: core</p> <p>IP Addresses: 192.168.220.104</p> <p>Host: esx-02b.corp.local</p>	

Figure 1.15 – FT loaded with the same IP on the host 02-b

As can be seen in Figure 1.15 the application server has the same IP on the secondary host without downtime

## Chapter 2

### Conclusion

“In the second chapter, we reviewed the problems of traditional networks and introduced Software-Defined network components.

We used vSphere to virtualize our servers and applications, and the result of this approach was High scalability. For example, to add another server and applications, we do not need to buy another server and spend lots of money on its hardware. We can use a pool of hardware resources that hypervisor provides, and in case of need, we can easily change the dedicated resource very fast. vCenter is central management, and by adding all hosts to the vCenter, we were able to configure all of them via a single management API, but in the traditional network, we needed to configure each server separately, and it was time-consuming.

We used different features that SDN provides in chapter two, features such as High Availability, which was a more common fault tolerance technique compared to FT, as we saw to have fault tolerance, we needed to have an identical pair on the other host. In other words, we needed to dedicate the ram, CPU, and other resources to an ideal server. Besides that, the configuration for FT was complicated, and all these reasons lead us to use High Availability for all applications and servers and only use FT for critical and essential servers.

Another important and useful feature that SDN gave to us was vMotion. With this feature, we were able to move a virtual server that was powered on to another cluster with all configuration; it is impossible to have a similar feature in traditional physical networks.”

# Chapter 3

## Network Virtualization with NSX-V

### Overview

In this chapter, after reviewing the difference between the standard switch and distributed switches and introducing virtual network components, we will install NSX-v in vCenter to have a centralized network management platform. We will use this management platform to create a universal distributed switch among all hosts to provide East-West connectivity. By using different port groups, we will separate network and management traffic. In the end, we will deploy a distributed logical router (DLR) to our hosts. Then we install and configure Edge routers to have different Network Function Virtualization for various purposes such as; NAT for providing connectivity between the outside and inside, Distributed logical routing (DLR), Distributed firewalling, and load balancing. In this chapter, we configure edges to have a dynamic interior gateway algorithm (BGP) and use distributed firewalls for deploying different policies.

### 3.1 Network Virtualization components

#### 3.1.1 Standard Switch (vSwitch)

The standard switch or sometimes called Vswitch is created by default when we install the ESXi hosts. It behaves like a regular physical switch inside each ESXi host and responsible for forwarding traffics generated by virtual machines inside the host to other virtual machines inside the same rack based on MAC address.

A standard switch is responsible for providing VMkernel access to networks for services like vMotion, iSCSI, FT and provides connectivity between:

different virtual machines within the same ESXi host. different virtual machines on different hosts physical and virtual machines on the network

Figure 3.1 depicted the architecture of Standard switch

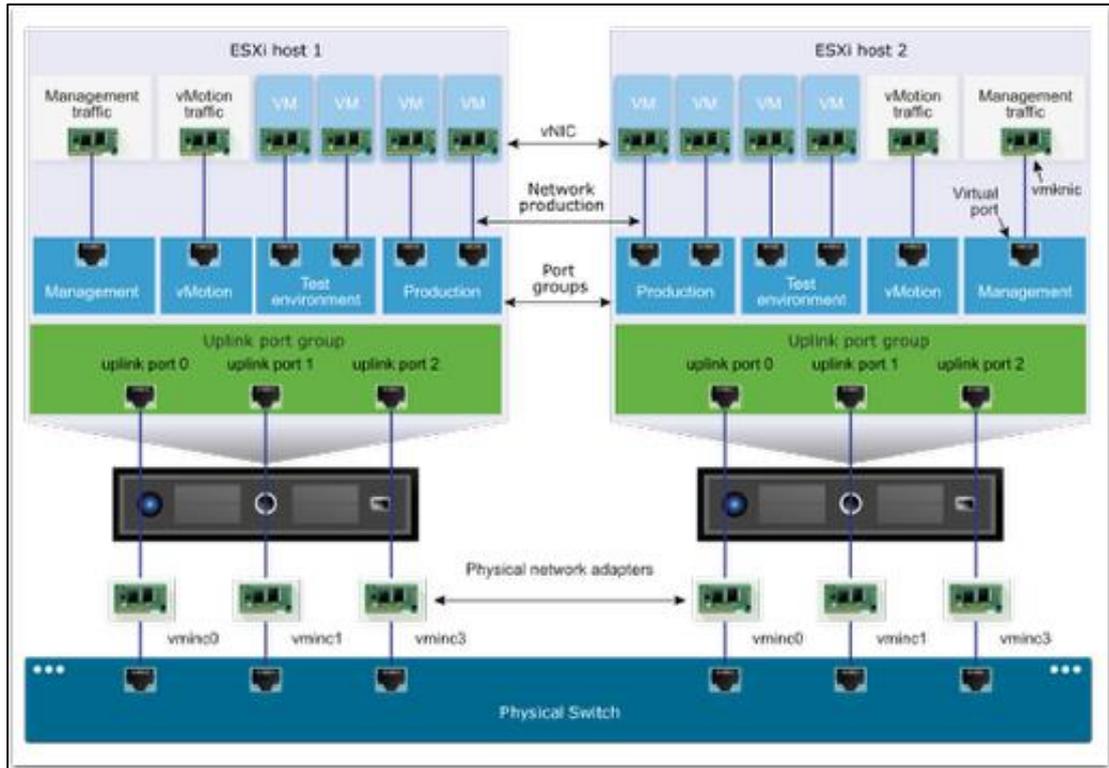


Figure 3.1- Architecture of Vswitch

Standard switch gives us the ability to provide layer two connectivity between VMs in the host internally. In other words, the virtual servers in the same subnet in the host can communicate directly, and traffic does not need to leave the ESXi host.

vSwitch also supports other advanced features like outbound traffic shaping, NIC teaming, Cisco Discovery Protocol (CDP), and different security policies.

### 3.1.2 Port Group

Port Groups is an aggregation of multiple ports under a common configuration to provide a stable point for VMs inside a specific labeled to communicate. We use port groups to separate VMkernel and server traffics.

### 3.1.3 Distributed Switch

Distributed switches provide connectivity between multiple hosts inside a cluster or different clusters. Creating a distributed switch and deploying different port groups gives us the flexibility to migrate virtual machines of a host to another host while they are in the same port groups.

Figure 3.2 depicted the architecture of a distributed switch.

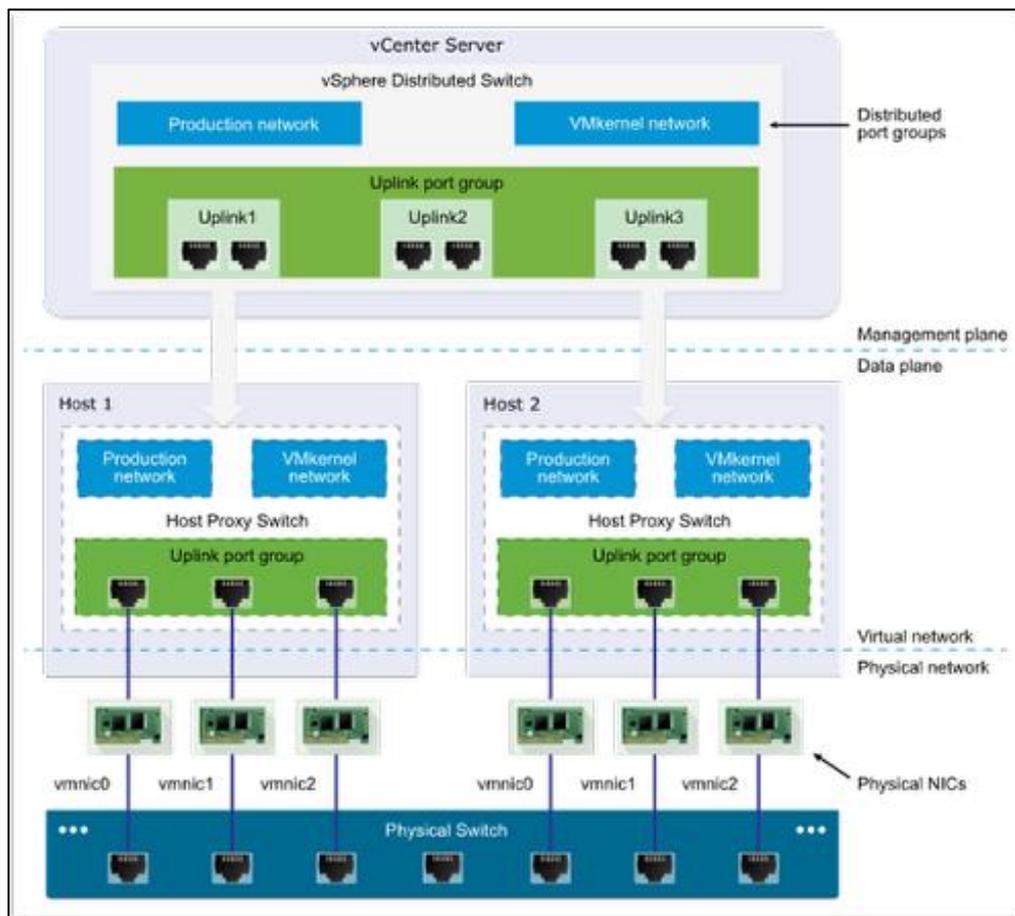


Figure 3.2 – Virtual Distributed Switch (Vds) architecture

As we discussed in chapter two, the best way of designing a Data Center is to separate computation and management clusters. The reason behind that is to ease

administrators troubleshooting and separating management and virtual machines traffic

#### VMNIC:

These ports are physical network interfaces adapter on the hosts, but NICs are virtual network interfaces.

### 3.1.4 VXLAN

VXLAN refers to Virtual Extension Lan. Before the invention of VXLAN, traditional Data Centers used VLAN to enforce layer two isolation. However, after passing the time and growing data centers, we needed to extend layer two networks across the racks inside a data center or even across data centers on different geographical locations. That was the reason for the VXLAN invention.

VXLAN is a solution for the extension of layer two networks over an existing layer three networks by encapsulating the whole frame with VTEP (Virtual Tunnel Endpoint) of outgoing host and decapsulating VTEP frame in the destination host.  
Figure 3.3

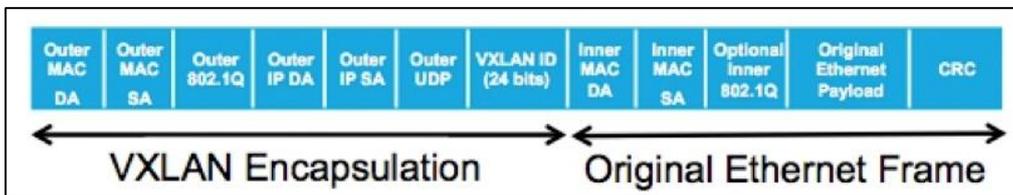


Figure 3.3 VXLAN frame format

### 3.1.5 Transport Zone

The transport zone is scope for VXLAN traffic. We can define several transport zones and virtual distributed switches and assign each of these Vds to a transport zone.

The recommendation for using the transport zone is to create a Global transport zone with a universal distributed switch that connects all the hosts and clusters in the data center and then creating other transport zones for different reachability scopes. When we create a transport zone; It gives us three different options for the control plane mode.

## Multicast

In this option, multicast sends traffic from a single source to several destinations. It requires PIM (Protocol Independent Multicasting) to be enabled as well as IGMP in the environment.

## Unicast

There is no need for specific physical network configuration like multicast, and the traffic is going through a single source to a single destination.

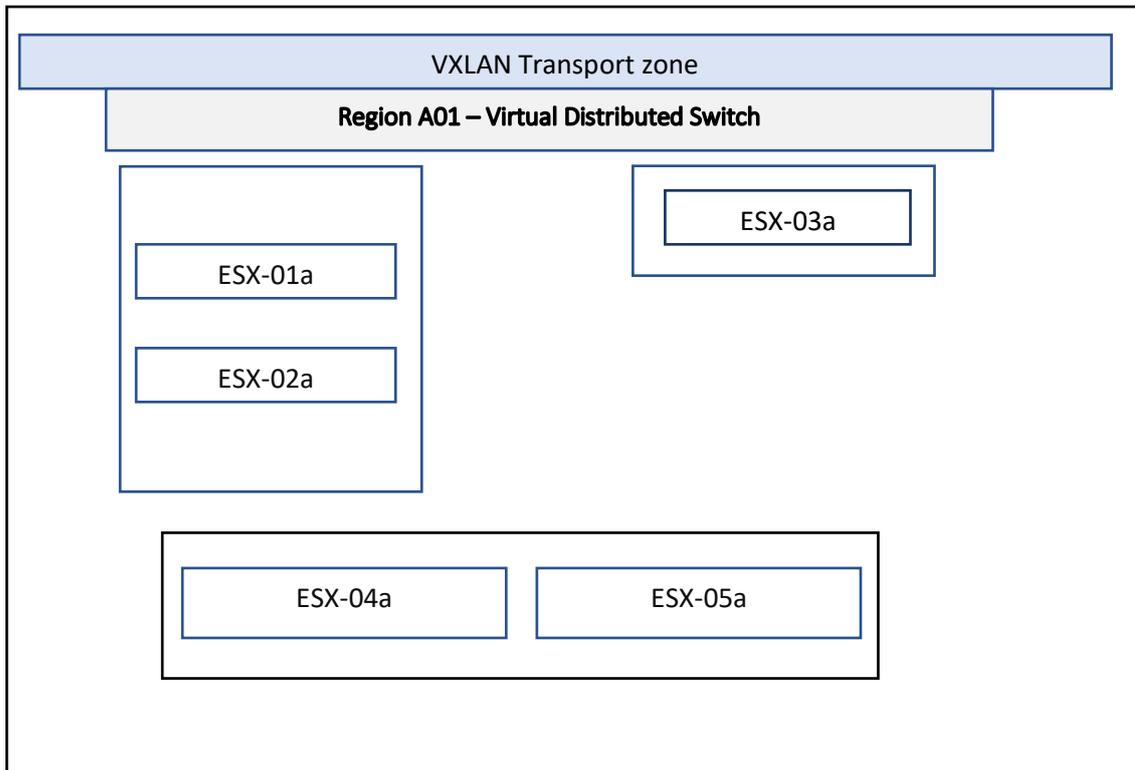
## Hybrid

It is a combination of unicast and multicast and IGMP needed on the first-hop switch, but we do not need PIM.

## 3.2 Virtual network architecture

We continue with the architecture we had in chapter one. In this step, we are going to create a global transport zone and a distributed switch, which includes all the hosts in the data center.

Figure 3.4 – Global architecture



Computational cluster:	Management cluster:
ESX-0a1 : 192.168.110.51	ESX-04a : 192.168.110.54
ESX-02a : 192.168.110.52	ESX-05a : 192.168.110.55
ESX-03a : 192.168.110.53	

As we discussed so far, to have a centralized network management platform, we need to install NSX in the vCenter. To deploy NSX into our architecture, we need to install the NSX manager first and exact three-nodes controller.

As depicted in figure 3.5, we installed the NSX manager inside the vCenter with an IP address of 192.168.110.42.

NSX Managers		NSX Controller Nodes			
ACTIONS ▾					
	NSX Manager	Role	IP	Version	vCenter
<input type="radio"/>	192.168.110.42	Standalone	192.168.110.42	6.4.5.13282012	vcasa-01a.corp.local

Figure 3.5

After installing the NSX manager, we install each controller separately—Figure 3.6

	Name	Controller Node	Status
<input type="radio"/>	controller-01	192.168.110.31 controller-1	Connected

Figure 3.6- fist controller installation

We installed NSX on vCenter and reached a central management point for network virtualization. To have connectivity between different hosts, we need to create a global transport zone to indicate the distributed switch scope. Then we should attach virtual machines that we need to have connectivity via this logical switch.

We created three virtual switches for web applications, Database applications, and transit switch. After that, we will attach virtual machines to relevant virtual switches. Figures 3.7, 3.8.

Name	Scope	Replication Mode	Logical Switches
RegionA0-Global-TZ	Global	Unicast	3

Figure 3.7 Universal Transport Zone

Logical Switch ID	Segment ID	Name	Status	Transport Zone	Connected VMs	Hardware Ports Binding	Scope	Replication Mode
virtualwire-3	5002	APP-LS	Normal	RegionA0-Global-TZ	3	0	Global	Unicast
virtualwire-2	5001	DB-LS	Normal	RegionA0-Global-TZ	3	0	Global	Unicast
virtualwire-1	5000	DC-Transit	Normal	RegionA0-Global-TZ	0	0	Global	Unicast

Figure 3.8 three logical switches that are attached to the Transport zone

We created three different logical switches for three different server categories in the scope of the transport zone, and then we attached three virtual machines to application and database logical switches.

## 3.3 Distributed Logical Router (DLR)

### 3.3.1 Overview

Routers responsible for routing traffic between different subnets. The problems of physical routers in traditional networking were not only the complexity of design when we scale-up, but also routers needed to process lots of east-west and north-south traffic.

DLR simplified this problem by distributing logical routers to hosts, and traffic does not need to leave the host for routing between different subnets; also, the routing is performing in the hypervisor kernel. Logical routing not only keeps the same functionality of physical routers but also optimizes handling east-west traffic.

Figure 3.9 illustrates the difference between routing in traditional physical networks and DLR.

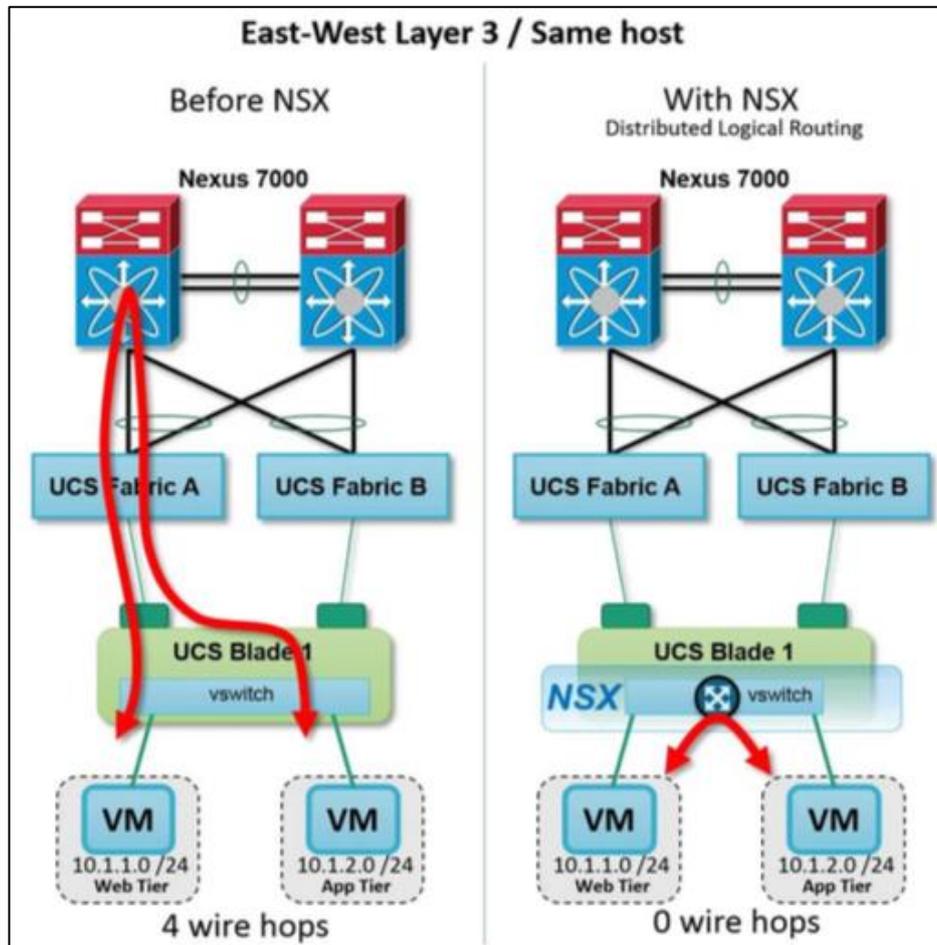


Figure 3.9 Difference between traditional network East-West routing and after using NSX

As we can see in figure 3.9, the packet in the web tier does not need to leave the ESXI host to receive the correct route and back again to the destination and waste bandwidth.

### 3.3.2 DLR Components

#### Data plane

The Data plane is a kernel module that is responsible for forwarding traffic. It consists of an NSX virtual switch with additional components for several services. Userspace agents, configuration files, and install scripts packaged in vSphere Installation Bundles (VIB) to run inside the kernel for providing different services such as DLR, firewalls, and enabling VXLAN bridging capabilities.

## Control plane

NSX Control plane consists of NSX logical router VM, Controller cluster, and user world agent.

Among all these components, the controller cluster is the most critical component, and it is responsible for:

- Eliminating ARP request broadcast traffic
- Distributing routing information to each host
- Performing load balancing between three NSX controller node
- Maintaining ARP tables
- Maintaining VXLAN and mac-address tables

## Management plane

The management plane is a central point of managing the whole NSX infrastructure, and its main features and responsibilities are:

- Installing VMware NSX controllers
- Configuring ESXi hosts via message bus agent
- Generating a self-signed certificate to have secure communication
- Installing UWA (User World Agent), Distributed logical routers and kernel modules for distributed firewall

NSX management has 1:1 relation with vCenter, and we can not service several data centers with one NSX management

A distributed logical router is an appliance that contains the control plane and distributed data plane in the kernel module of each hypervisor. DLR control plane needs to communicate with the NSX controller to push all routing updates from the management plane to the kernel modules. The interaction between DLR and control cluster to push updates to the NSX logical router is shown in [Figure 3.9](#)

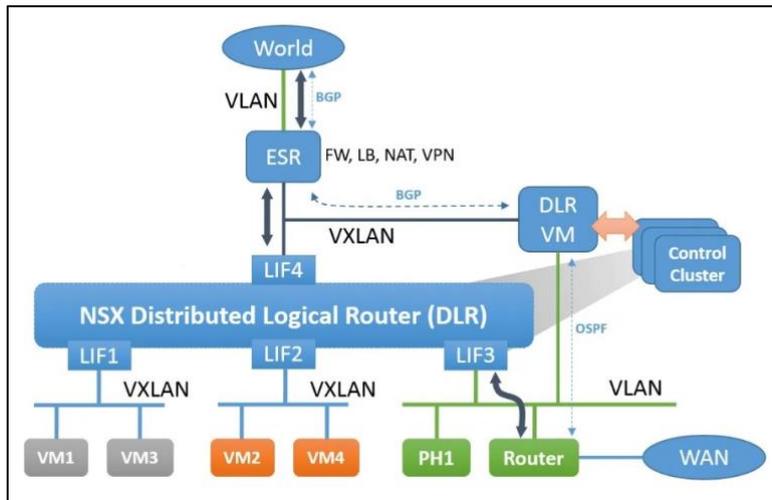


Figure 3.9 – Interaction between DLR VM and control cluster

### 3.4 Deploying DLR to architecture

In this section, we are going to separate virtual servers and applications by their category into three logical switches in the scope of transport zone, and then we will use DLR to have a distributed logical route between these three virtual switches.

We created logical switches for web servers, applications, databases, and DC transit. Then we deployed each virtual machine to associate logical switch. Figure 3.10

Logical Switch ID	Segment ID	Name	Status	Transport Zone	Connected VMs
virtualwire-3	5002	APP-LS	Normal	RegionA0-Global-TZ	3
virtualwire-2	5001	DB-LS	Normal	RegionA0-Global-TZ	3
virtualwire-1	5000	DC-Transit	Normal	RegionA0-Global-TZ	0
virtualwire-4	5003	WEB-LS	Normal	RegionA0-Global-TZ	2

Figure 3.10 logical switches

As we can see inside the logical switch, virtual machines can ping each other, but they do not have any feasible route to other subnets. Figure 3.11, 3.12

```
Welcome to Photon 1.0 (x86_64) - Kernel 4.4.8-esx (tty1)
app-01a login: root
Password:
root@app-01a [ ~ ]# ping 172.16.10.11
PING 172.16.10.11 (172.16.10.11) 56(84) bytes of data.
-
```

Figure 3.11 – Application servers can not ping web application

```
root@web-02a [ ~ ]# ping 172.16.10.11
PING 172.16.10.11 (172.16.10.11) 56(84) bytes of data.
64 bytes from 172.16.10.11: icmp_seq=1 ttl=64 time=3.27 ms
64 bytes from 172.16.10.11: icmp_seq=2 ttl=64 time=0.534 ms
```

Figure 3.12 Inside the switch they can ping

To have a route between different subnets, we need to deploy a Distributed Logical Router (DLR) and assign an IP address in the same range of each virtual switch. Figure 3.13

vNIC#	Name	Type	IP Address	Connected To	Connection Status
2	DC-Transit	Uplink	172.16.0.10/24	DC-Transit	Connected
10	DB-LS	Internal	172.16.30.1/24	DB-LS	Connected
11	APP-LS	Internal	172.16.20.1/24	APP-LS	Connected
12	web-LR	Internal	172.16.10.1/24	WEB-LS	Connected

Figure 3.13 – DLR interfaces which connected to each logical switch

After that, all these logical switches connected to a logical router and different subnets have a connection interface to the logical router so they will be able to send and receive ICMP packets.

```
root@web-02a [ ~ ]# ping 172.16.20.11
PING 172.16.20.11 (172.16.20.11) 56(84) bytes of data.
64 bytes from 172.16.20.11: icmp_seq=1 ttl=63 time=929 ns
64 bytes from 172.16.20.11: icmp_seq=2 ttl=63 time=0.758 ms
```

So far what we deployed is illustrated in figure 3.13

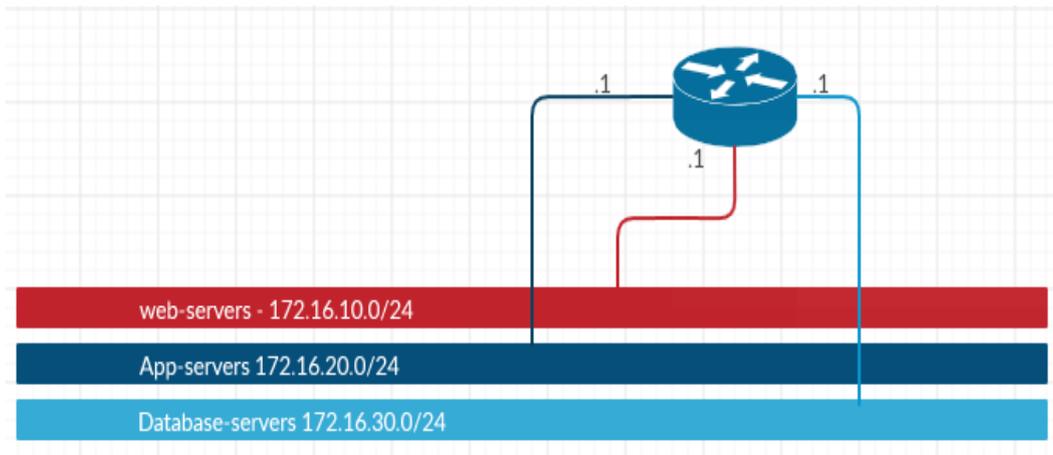


Figure 3.13

## 3.5 Deploying Dynamic routing algorithm on DLR

### 3.5.1 NSX Edge Gateway

NSX edge is an appliance that is responsible for providing connectivity and handling traffic that is going outside and coming inside. It acts as a router and able to peer with physical network equipment, and gives the network virtualization access the internet, WAN, or any other physical network resources.

### 3.5.2 Edge Service Gateway

ESG gives us access to many services such as; FIREWALL, DHCP, VPN, and Load balancing. We can install a cluster of Edges to have redundancy and not having a single point of failover.

In this section, we will deploy Edge service gateway, and by configuring border gateway routing protocol on the edge router, we will provide connectivity between inside network and outside network.

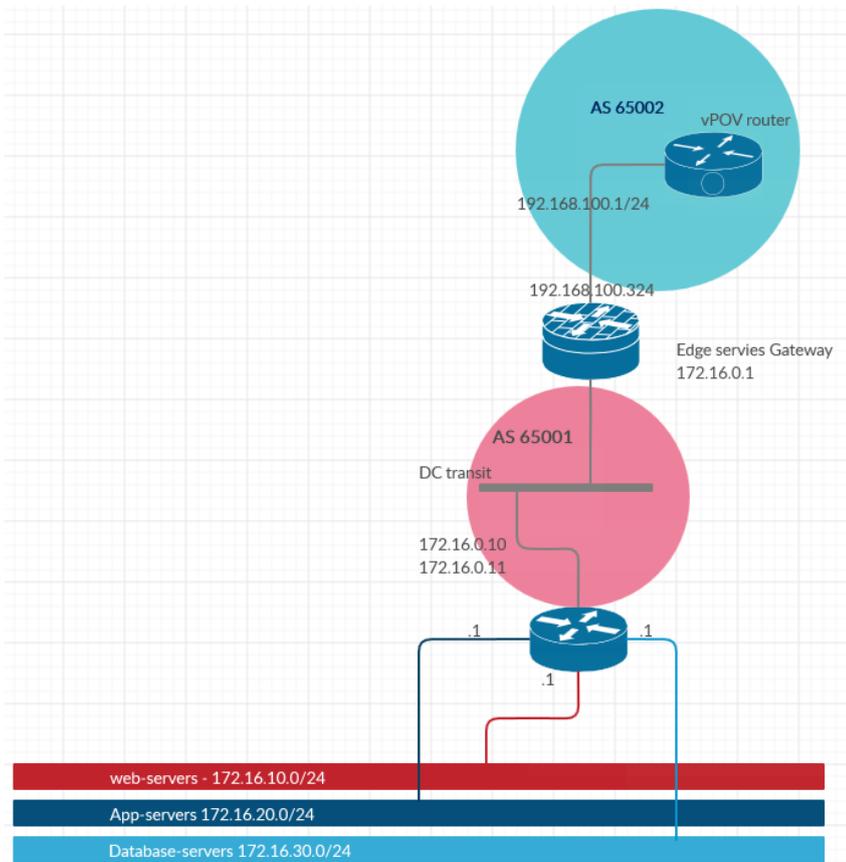


Figure 3.14

### 3.5.3 Deploying BGP protocol

In this section, we will deploy the BGP routing protocol to the edge router to provide north-south connectivity.

First, we set the uplink interface of DLR to 172.16.0.10; for simplicity, we also set the router ID of this router with this IP.

As we can see in Figure 3.14, the default gateway is 172.16.0.1. Figure 3.15

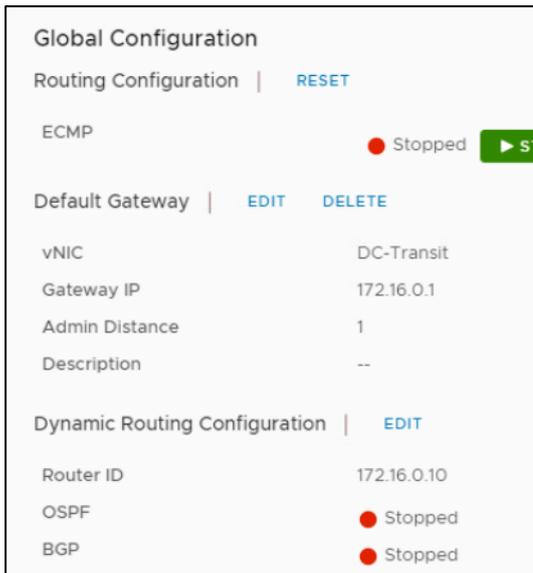


Figure 3.15

We saved these changes and published them to or DLR.

In the second step, we should enable the border gateway protocol. As we can see in our design, we have one interface of BGP in DC-transit with an autonomous system id of 65001 and the following addresses.

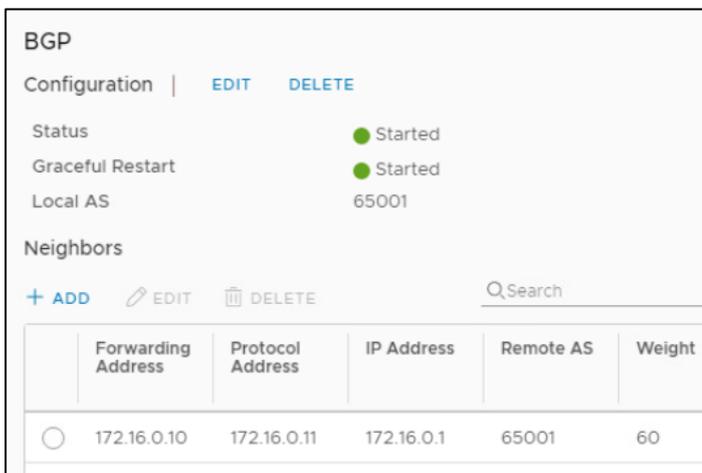


Figure 3.16

Route redistribution disabled by default; to activate it and to allow the BGP to learn connected routes and permit packets to pass, we need to configure it in the DLR route redistribution tab. Figure 3.17

Prefix Name	Any
Learner Protocol	BGP
Allow Learning From	<input type="checkbox"/> OSPF <input type="checkbox"/> BGP <input type="checkbox"/> Static Routes <input checked="" type="checkbox"/> Connected
Action	Permit

Figure 3.17

The final step for this section is to configure the NSX EDGE to have one interface to BGP autonomous 65001 and one interfaces as an uplink to autonomous 65002

**BGP**

Configuration | [EDIT](#) [DELETE](#)

Status ● Started

Graceful Restart ● Started

Default Originate ● Stopped

Local AS 65001

**Neighbors**

[+ ADD](#) [EDIT](#) [DELETE](#)

	IP Address	Weight	Keep Alive Time (Seconds)	Hold Down Time (Seconds)
<input type="radio"/>	172.16.0.11	60	60	180
<input type="radio"/>	192.168.100.1	60	60	180

Figure 3.18

As we can, NSX Edge has two neighbors in two different subnets. Now we configured everything we need to have north-south IP routes and connectivity.

To verify our configuration, we will use one of the PC on range 192.168.110.1/24 to reach a web server, which has an IP address of 172.16.10.11.

```

IPv4 Address . . . . . : 192.168.110.10
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : fe80::250:56ff:fe01:f5b1%12
                          192.168.110.1

Tunnel adapter isatap.{F5453641-443F-4B54-9BDE-B6FB4E2E741C}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . : corp.local

C:\Users\Administrator>ping 172.16.10.11

Pinging 172.16.10.11 with 32 bytes of data:
Reply from 172.16.10.11: bytes=32 time=1ms TTL=61
Reply from 172.16.10.11: bytes=32 time=1ms TTL=61
Reply from 172.16.10.11: bytes=32 time=1ms TTL=61

```

Figure 3.19

## 3.6 Distributed Firewall and micro-segmentation

### Overview

NSX Distributed Firewall (DFW) is a kernel-embedded firewall in the hypervisor of hosts that control and provides visibility for virtualized workloads gives us the ability to create policy based on IP, IPsec, VXLAN, VLAN, virtual machines, clusters, datacenter.

As the firewall is distributed to each VM when we create a policy and deploy it to the network topology, these rules will force at the VNIC level of each virtual machine. Then after vMotion or HA, these rules remain without any changes. This nature of distributed firewalls has significant benefits compared to physical firewalls in the traditional network, which is an automatic extension by adding hosts or virtual machines.

Microsegmentation is a technique in network security, and by this technique, the network architects able to logically divide into separate security and deliver services for each segment.

### 3.6.1 Service composer

Service composer lets us build a new model for consuming network and security services and assigning these security services and policies to applications in real-time.

In this section, we will change the default firewall policy for deploying different policies. As we checked in figure 3.19, we have connectivity to different servers. Now we want to create a specific firewall rule for web applications.

In the composer section, we create a web-tier group and join two web applications to this group as illustrated in Figure 3.20

Name	Applied Security Policies
Activity Monitoring Data Collection	4
web-tier	0

Figure 3.20

In the firewall configuration, we are going to create three different rules

- **Exit to web:** all traffic from any source in which destination is for web-tier and using HTTPS service have permitted to pass traffic.
- **Web to the app:** all traffic is coming from web servers and destined to application logical switch, which consists of application servers, and using customized service of MY-APP (protocol TCP with destination port 8443) should have the permission to pass.
- **All traffics** coming from logical application switch, consists of application servers and destined to Database logical switch and using HTTP service, should be allowed to pass traffic.

We created these rules on the firewall. Figure 3.21

#	Name	ID	Source	Destination	Service	Applied To	Action	Log
1	EXIT-WEB		Any	web-tier	HTTPS	Distributed Fir...	Allow	<input checked="" type="checkbox"/>
2	WEB to APP		web-tier	APP-LS	MY APP	Distributed Fir...	Allow	<input checked="" type="checkbox"/>
3	app to DB		APP-LS	DB-LS	HTTP	Distributed Fir...	Allow	<input checked="" type="checkbox"/>

Figure 3.21 – Firewall rules

The default rule is for any sources that send traffic to any destination to use any service is set to deny or block. Then, by creating access permission to these groups, we create a high-security level for our systems.

We expect the firewall drop ICMP packets when we ping three different subnets, and as we can see in figure 3.22, all packets dropped. The reason is we did not define any permission rule for ICMP protocol from any sources, and the default rule of firewall, which configured to reject anything else, is discarding our packets at the kernel level of each host.

```
root@web-01a [ ~ ]#  
root@web-01a [ ~ ]#  
root@web-01a [ ~ ]#  
root@web-01a [ ~ ]# ping -c 2 172.16.10.12  
PING 172.16.10.12 (172.16.10.12) 56(84) bytes of data.  
  
--- 172.16.10.12 ping statistics ---  
2 packets transmitted, 0 received, 100% packet loss, time 1471ms  
  
root@web-01a [ ~ ]# ping -c 2 172.16.20.11  
PING 172.16.20.11 (172.16.20.11) 56(84) bytes of data.  
  
--- 172.16.20.11 ping statistics ---  
2 packets transmitted, 0 received, 100% packet loss, time 1183ms  
  
root@web-01a [ ~ ]# ping -c 2 172.16.30.11  
PING 172.16.30.11 (172.16.30.11) 56(84) bytes of data.  
  
--- 172.16.30.11 ping statistics ---  
2 packets transmitted, 0 received, 100% packet loss, time 1190ms
```

Figure 3.22 – Firewall is dropping ICMP packets from web servers to any other subnets

## Chapter 3

### Conclusion

“In this chapter, we focused on network virtualization with NSX-V. NSX-v gave us the ability to manage our network infrastructure centrally.

As we discussed, it used three nodes controller to push applied configuration in the management layer to forwarding nodes. In virtual networks, we should have the same functionality we had in physical networks. NSX-V not only provided this functionality but also gave us the flexibility, programmability, and elasticity to have a virtual switch with many more ports we could have in physical switches. We had at most 48 ports in real physical switches, but we could have 1061 virtual ports in virtual switches.

The second benefits that we reached were in virtual switches; it did not need to send out traffic host to the physical network and use bandwidth, and it could be sent traffic to another application in the same host (rack).

Besides, virtual switches, we had distributed logical routers. DLR was a kernel module logical router and provided routes between different logical switches internally. For example in physical networks to send traffic from subnet A to subnet B we had to use a physical router and traffic had to reach this router to receive the route, as we know routers are expensive, and network administrators had to send their request to t company and get their confirmations to buy a router, and which was time-consuming. In DLR, we can easily implement a router immediately, and the traffic does not need to leave the host.

Another essential concept we had in virtual networks was that we were able to create a universal switch between applications and servers in the same host and between a different host in different places.”

## Chapter 4

# NFV and NSX-T

## Overview

VMware NSX-T is another software-defined network tool to virtualize the network infrastructure, and the main difference of this type in comparison to NSX-V is, it can support multi-hypervisor environments such as KVM or multiple public/private clouds.

In this chapter, after reviewing components of NSX-T, we are going to add a KVM hypervisor and use NSX-T to deploy a logical switch between VMware hypervisor and KVM. After that, we will use features that NSX-T provides, such as NAT to convert public IP and private IP, Firewall to deploy different policies, and load balancing to provide redundancy and failover.

### 4.1 NSX-T components

#### Management plane

NSX-T manager is the core of the management plane, and it gives administrators a comprehensive view of everything involved via a single API entry point.

#### Control plane

The control plane is responsible for pushing and distributing information from the management plane to forwarding engines. This control plane composed of two parts. Figure 4.1

- The central control plane (CCP): Use a cluster of virtual machines called CCP to have a redundant. Any failure in CCP is not affecting the data plane, and data is not passing through this level
- The local control plane (LCP): it runs in transport nodes and communicate with CCP

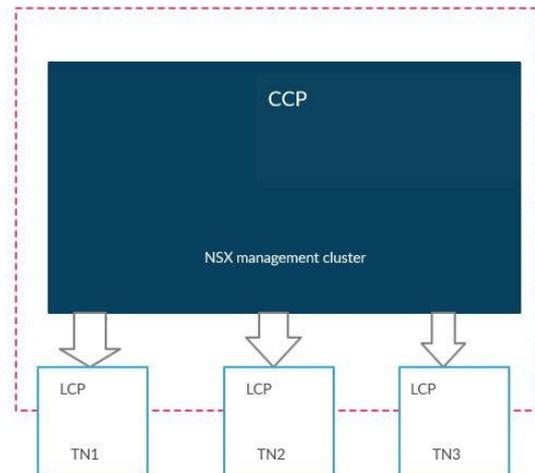


Figure 4.1

### Data Plane

Data plane is the place that all stateless forwarding takes place. In this level, we have transport nodes or N-VDS, which are unique for NSX-T.

### Hypervisor Transport nodes

These are hypervisors that prepared to run NSX-T; they can be VMware hypervisor (ESXi) or Linux base hypervisor (KVM).

### N-VDS

N-VDS is a software define switch platform which is entirely hypervisor independent. N-VDS's responsibility is forwarding traffic among components of transport nodes or between internal components and underlay physical network.

### EDGE nodes

EDGE nodes are server appliance, and their primary jobs are doing services that are not dispersed among any hypervisors (ESXI or KVM). They are representing pools of capacity and can be grouped in a cluster.

### **Hypervisor transport nodes**

They are ESXi or KVM hypervisors that participate in NSX-T.

### **UPLINK profile**

Uplink profiles are templates to define how N-VDS can connect to the physical network, but also, they ensure the profile is applied uniformly across multiple transport nodes.

In uplink profile we can specify:

- Uplink MTU
- Uplink format of N-VDS
- Teaming policies for the uplinks

We are going to add a Linux hypervisor (KVM) to the network and then use NSX-T centralized management API to deploy different services such as NAT, Load balancing, Firewalling to this multi-hypervisor infrastructure.

## **4.2 The virtual Network architecture in multi-hypervisor**

We added a KVM hypervisor and two NSX-edge clusters to the previous design

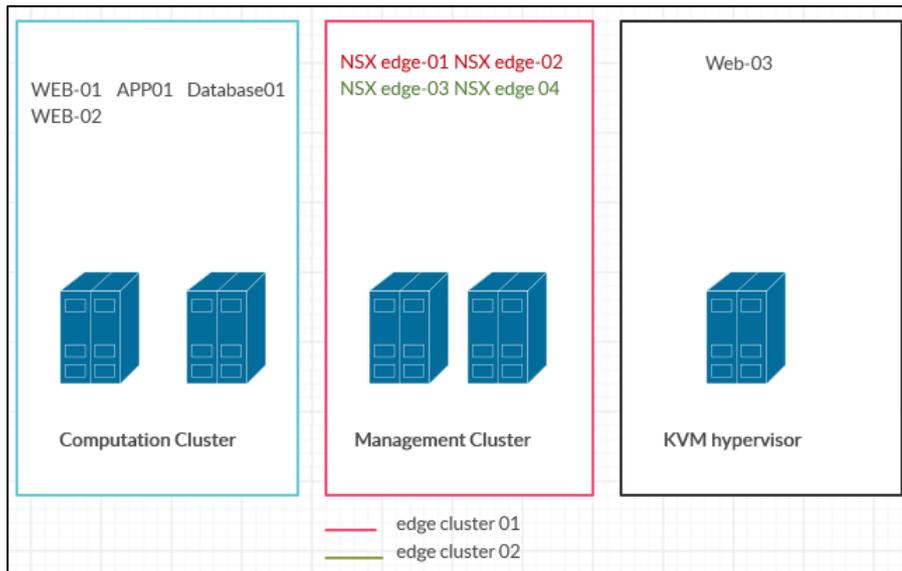


Figure 4.2

After clustering our virtual infrastructure, we designed a two-tier gateway architecture.

### Tier one

Tier one is a gateway router which connected to tier zero for northbound traffic routing in one hand and on the other hand connected to one or more underlay networks

### Tier zero

Tier one is a top tier layer, and on the one hand, it has an interface to tier one, on the other hand, it has an interface to another network. We can deploy tier zero as an active-active or active-standby cluster.

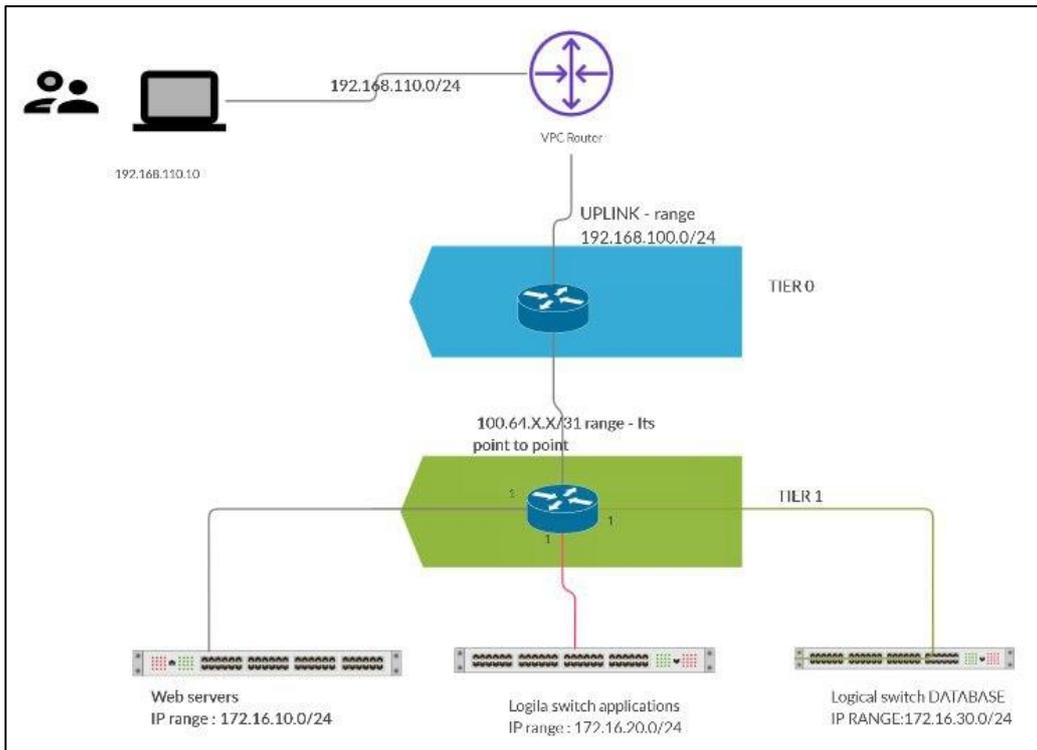


Figure 4.3

We added KVM to our design, and as we can see in the configuration of host transport nodes in Figure 4.4, hosts that are running VMware hypervisors are managed by vCenter, but the KVM is a standalone host.

Host Transport Nodes		Edge Transport Nodes	Edge Clusters	
Managed by vCenter				
<input type="button" value="CONFIGURE NSX"/> <input type="button" value="REMOVE NSX"/> <input type="button" value="ACTIONS"/>				
<input type="checkbox"/>	Node	ID	IP Addresses	Os Type
<input type="checkbox"/>	RegionA01-MG...	MoRef ID: ...		
<input type="checkbox"/>	esx-03a.corp.local	8bcc...t-32	192.168.110.53, 10.10.20.53, ...	ESXi 6.7.0
<input type="checkbox"/>	esx-04a.corp.local	8bcc...t-33	192.168.110.54, 10.10.20.54, ...	ESXi 6.7.0
<input checked="" type="checkbox"/>	RegionA01-CO...	MoRef ID: ...		
<input type="checkbox"/>	esx-01a.corp.local	be42...aa6b	192.168.110.51, 10.10.20.51, ...	ESXi 6.7.0
<input type="checkbox"/>	esx-02a.corp.local	3195...dda8	192.168.110.52, 10.10.20.52, ...	ESXi 6.7.0

Managed by <span>None: Standalone Hosts</span>				
<a href="#">+ ADD</a> <a href="#">EDIT</a> <a href="#">DELETE</a> <a href="#">ACTIONS</a>				
<input type="checkbox"/>	Node	ID	IP Addresses	OS Type
<input type="checkbox"/>	<a href="#">kvm-01a.corp.local</a>	5ba9...3c18	192.168.110.61	Ubuntu ...

Figure 4.4 – host transport nodes configuration in a multi-hypervisor environment

We create a new logical segment for the added KVM host, we name it "newsegment" and attached host web-04 to this logical segment.

	Segment Name	Uplink & Type
>	<a href="#">LS-app</a>	Tier-0-gateway-01   Tier0 - Flexible
>	<a href="#">LS-db</a>	Tier-0-gateway-01   Tier0 - Flexible
>	<a href="#">LS-web</a>	Tier-0-gateway-01   Tier0 - Flexible
>	<a href="#">Newsegment</a>	Tier-0-gateway-01   Tier0 - Flexible
>	<a href="#">Uplink</a>	None - Flexible

Figure 4.5

All these virtual switches are in different subnets, but all of them have an interface uplink connectivity to tier 0 gateway, and we expect all of these web servers from the different hypervisors and subnets to be able to send and receive ICMP packets.

```

C:\Users\Administrator>ping web-01a

Pinging web-01a.corp.local [172.16.10.11] with 32 bytes of data:
Reply from 172.16.10.11: bytes=32 time=8ms TTL=62
Reply from 172.16.10.11: bytes=32 time=1ms TTL=62

Ping statistics for 172.16.10.11:
    Packets: Sent = 2, Received = 2, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 8ms, Average = 4ms
Control-C
^C
C:\Users\Administrator>ping web-02a

Pinging web-02a.corp.local [172.16.10.12] with 32 bytes of data:
Reply from 172.16.10.12: bytes=32 time=6ms TTL=62
Reply from 172.16.10.12: bytes=32 time=1ms TTL=62
Reply from 172.16.10.12: bytes=32 time=1ms TTL=62
Reply from 172.16.10.12: bytes=32 time=1ms TTL=62

Ping statistics for 172.16.10.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 6ms, Average = 2ms

C:\Users\Administrator>

C:\Users\Administrator>ping web-03a

Pinging web-03a.corp.local [172.16.10.13] with 32 bytes of data:
Reply from 172.16.10.13: bytes=32 time=12ms TTL=62
Reply from 172.16.10.13: bytes=32 time=1ms TTL=62

```

Figure 4.6

## 4.3 NAT and DHCP with NSX-T

In this section, we will complete our architecture by adding the second tier, which is tier one. Tier one is connected to tier zero and use the second edge cluster. After that, we will add a web server logical switch to this tier, and our goal is to provide a source and destination NAT for these servers.

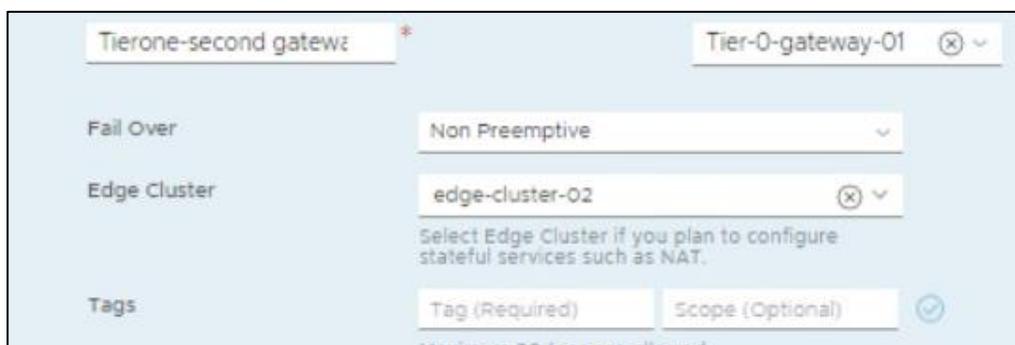


Figure 4.7- Adding the second tier which is tier-one to connect tier 0 by using edge cluster 02



Figure 4.8 – We connected web server logical switch to tier-one

NAT configured to translate source with 172.16.10.0/24 to 80.80.80.0/24 as source NAT, and any source with the destination IP address of 80.80.80.0/25 to 172.16.10.13 (web-03 IP address). Figure 4.5

>	web-03	SNAT	172.16.10.13	Any	80.80.80.1	Up
>	web-03	DNAT	Any	80.80.80.1	172.16.10.13	Up

Figure 4.5 – Source and destination NAT for web-03

Then we need to redistribute this NAT in the tier-one configuration to let these translated traffic pass through it.

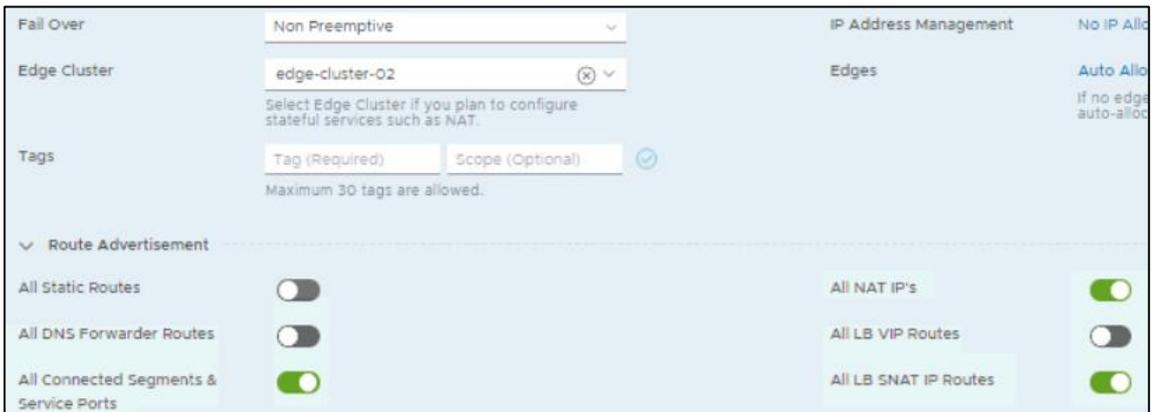


Figure 4.6 – NAT route advertisement

To verify this configuration, we can ping 80.80.80.1, and we expect to have successful ping without any packet loss, and with tracert command, we can monitor the path which packet passed to reach webserver.

```

C:\Users\Administrator>ping 80.80.80.1

Pinging 80.80.80.1 with 32 bytes of data:
Reply from 80.80.80.1: bytes=32 time=22ms TTL=61
Reply from 80.80.80.1: bytes=32 time=2ms TTL=61
Reply from 80.80.80.1: bytes=32 time=1ms TTL=61
Reply from 80.80.80.1: bytes=32 time=2ms TTL=61

Ping statistics for 80.80.80.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 22ms, Average = 6ms

C:\Users\Administrator>tracert 80.80.80.1

Tracing route to 80.80.80.1 over a maximum of 30 hops
  0  <1 ms    <1 ms    <1 ms    router-110.corp.local [192.168.110.1]
  1  <1 ms    <1 ms    <1 ms    192.168.100.3
  2  *         *         *         Request timed out.
  3  4 ms     1 ms     1 ms     80.80.80.1

```

Figure 4.6 – verifying the source and destination NAT

This successful ping means the both Source NAT and destination NAT is working well, and router gateway lets the traffic to pass.

## 4.4 Dynamic host configuration protocol (DHCP protocol in NSX-T)

Dynamic host configuration protocol is a protocol that gives the ability to end-users to receive their configuration automatically without the need of an administrator's interaction.

The dynamic automation is beneficial for reducing the time and misconfiguration of user admins, especially when there are many end hosts to configure.

In this section, we will add a DHCP server to our design by dedicating a segment for the DHCP server and then attach it to the tier-zero gateway. After that, we need to introduce this server to logical switches. By doing this, our end hosts will receive their IP address and configuration from the DHCP server automatically.

In the first step, we should create a DHCP server and allocate a static IP to it. Figure 4.7

Choose either local or remote DHCP or No Dynamic IP Allocation.

Type	DHCP Local Server	
DHCP Server	DHCP-SERVER	
Server Details	Server Name	DHCP-SERVER
	Lease Time	86400 seconds
	Server Address	172.16.60.1/24

Figure 4.7

After that, we create a logical segment for the DHCP server and attach it to Tier-zero

DHCP-SERVER-LS *	Tier-0-gateway-01   Tier0
------------------	---------------------------

Figure 4.8

Then in the IP management of the DHCP segment that we created, we need to introduce the gateway and range of IP we want to assign for the end hosts (we want to receive IPs between range (172.16.60.10 – 172.16.60.30). Figure 4.8

Gateway	DHCP Ranges
<input type="text" value="172.16.60.1/24"/> * <small>Format CIDR e.g. 10.12.2.1/24</small>	<input type="text" value="172.16.60.10-172.16.60.30"/> X Enter DHCP Ranges

Figure 4.8

Now, if we attach any end host to this segment, we expect to take an IP address in this range automatically. To verify that, we attach web-04 to this segment. Figure 4.9

```
eth0      Link encap:Ethernet  HWaddr 00:50:56:a5:51:ba
          inet addr:172.16.60.14  Bcast:172.16.60.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:fea5:51ba/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:36 errors:0 dropped:0 overruns:0 frame:0
          TX packets:488 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2670 (2.6 KB)  TX bytes:21923 (21.9 KB)
```

Figure 4.9 Web 04 received an IP range we allocated in DHCP

## 4.5 Load balancing in NSX-T

### Overview

The load balancer is responsible for distributing the incoming traffic from users across multiple or group of backend servers.

In this section, in the first step, we will create a load balancer and second tier-one gateway router, and then we will connect one of the interfaces to load balancer and the second one to the tier-one router that we created in the previous section.

The goal of this section is to provide load balancing and failover for web servers in our data center, and in the case of the host failure, the second web server service our customers immediately. Also, we will create an HTTPS monitoring to see how all the processes work.

webapplication_monitor	HTTPS	443	5	15
------------------------	-------	-----	---	----

We start this section by creating a secondary tier-one gateway and connecting it to tier-zero gateway. Figure 4.10

	Tier-1 Gateway Name	Linked Tier-0 Gateway
>	tier_to_loadbalancer	Tier-0-gateway-01
>	tierone-second gateway	Tier-0-gateway-01

Figure 4.10

Then we need to allow all NAT and load balancing traffic to pass by changing route redistribution configuration.

Route Advertisement	
All Static Routes	<input checked="" type="checkbox"/>
All DNS Forwarder Routes	<input checked="" type="checkbox"/>
All Connected Segments & Service Ports	<input checked="" type="checkbox"/>
All NAT IP's	<input checked="" type="checkbox"/>
All LB VIP Routes	<input checked="" type="checkbox"/>
All LB SNAT IP Routes	<input checked="" type="checkbox"/>

Figure 4.11

Then we need to allow route redistribution of virtual IP in the tier zero. After that, we create a pool for servers, which is known as a server farm, to have load balancing on them. In this case, we need to use a pool of web servers. We choose the round-robin algorithm.

The round-robin algorithm gives us the ability to send traffic to the first gateway and the second traffic to the second gateway.

Name	IP	Port	Weight	State	Backup Member	Max Concu Conne
web02a	172.16.0.2	443	1	Enabled	● Disabled	
web-01a	172.16.0.11	443	1	Enabled	● Disabled	

Figure 4.12- web server pool

At this step, we need to configure a virtual server. The reason behind introducing a virtual server is that by using a single IP and introducing it to all clients or hosts, we can aggregate all requests in a single point and then distribute them among servers. For this reason, we will create a virtual server and set it with an IP address of a load balancer.

As depicted in figure 4.12, we created a virtual server with the name of VIP, and then we introduced a single VIP to all clients, round-robin cycle, and sends traffic that is coming for protocol 443(HTTPS) to webserver pool which we created. Figure 4.12

Name	IP Address	Ports	Type	Load Balancer	Server Pool
Webapp-VIP	172.16.10.10	443	L4 TCP	load_balancer	Webserver-pool

Figure 4.13 virtual server in NSX-T

We send a request to the web server, and as we can see in figure 4.13, we accessed via web-02.

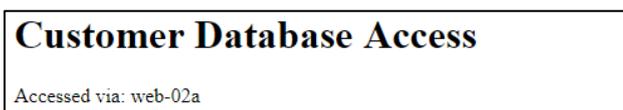


Figure 4.14

Now, if we will turn off the web server02 or send many requests quickly, we can see the load balancer works well and will receive service via web server web-01.



Figure 4.15

## Chapter 4

### Conclusion

“In chapter four, we discussed NSX-T, which has all functionality of NSX-V and also supports not only multi-hypervisor environments but also supports public/private clouds. We deployed three features that NSX-T provides, NAT, load balancer, and firewalls.

NAT:

Network Address translator invented as a solution for the shortage of IPv4, and it converts two different IPs. We usually use this mechanism to translate bidirectionally private IP to public IP. In this chapter, we used NAT at the edge of the design to convert private IPs inside the data center to public IP for having connectivity to the internet and external environment.

Load balancing:

To distribute loads of the system, we used load balancing, as we expect to have many requests from the user side to web servers, the best practice is to distribute the loads and do not have a single point of failure.

Firewall:

Firewalls are essential networks component, both in traditional physical and virtual networks. The difference between logically distributed firewalls and traditional is, DFW is distributed in kernel modules, and the generated traffic does not need to reach the firewalls to receive an appropriate action (permit, block, restrict), and the kernel module can decide it. The second benefit of DFW is flexibility and scalability. We are not limited to just source and destination, services, or port numbers. We are able to have different rules for different virtual machines in the same host.”

## **Chapter 5**

# **vRealize Network Insight**

## **Overview**

VMware Network Insight is another VMware product that brings intelligent operations for SDN and enables global view across both virtual and physical networks and gives administrators the ability to find pinpoint of the problem in a complex network and optimize security and firewall rules. Moreover, it can help us improve performance and availability by converging all physical and virtual network information, and it is available for on-premises sites or clouds. In this chapter, we are going to monitor the whole infrastructure and track a connection between two different hosts and see how vRealize can help us to track connections; then, at the end of the chapter, we will use application discovery to create an application automatically with all its connections.

### **5.1 Traffic distribution**



Insight appliance and monitor the real-time traffic patterns and use its firewalls and security recommendation for added nodes.

The East-West section in figure 5.1 shows the sum of all traffic flows with the percentage as East-West traffic, by going through this section we can see all traffic and flows for 16-17 June 2020.

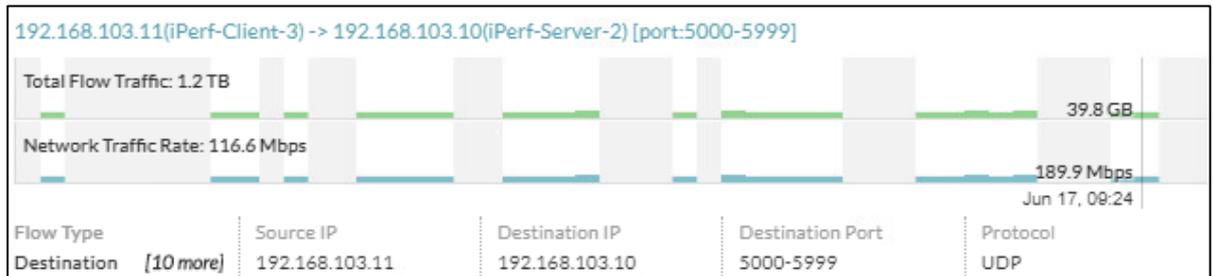


Figure 5.2

On the top ports of figure 5.1, we open one of the top ports to analyze, we will open port 336, and as we can see in figure 5.3, it is the traffic that is passing from one of the webservers to database servers on TCP protocol and using port 3306.

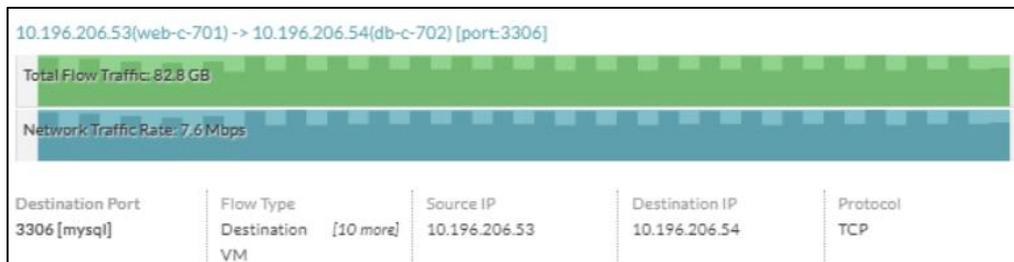


Figure 5.3

If we go more in details for this flow, we can see the source, destination, protocols, IPs, and firewall rules

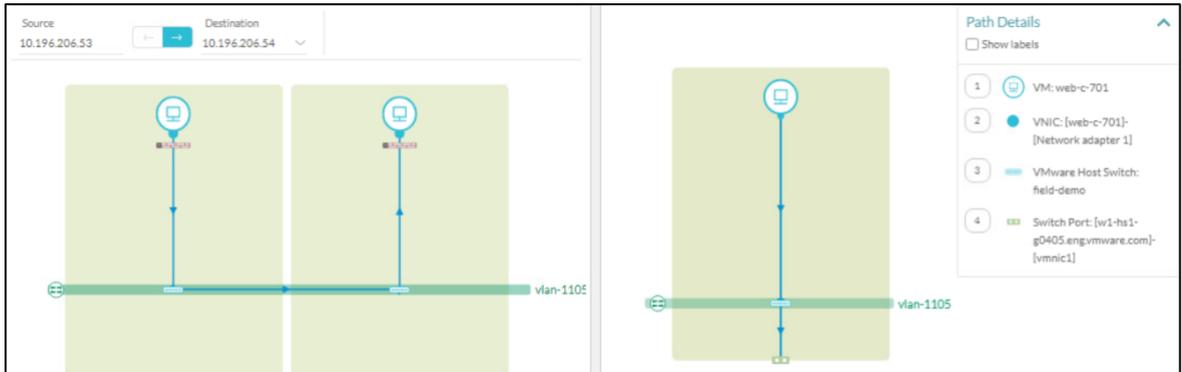


Figure 5.4- network topology flow path between web server source and database destination

NSX Firewall							
Name NSX Firewall		Total Rule Count 3	Manager wdcnsx-master.cmbu.local				
Firewall Rules Showing 3 (view all)							
Seq ID	Name	Configured Source	Configured Destination	Service	Action	Section Name	
3	Default Rule NDP	Any	Any	IPv6-ICMP Neighbor Advertisement or Solicitation	ALLOW	Default Section Layer3	
4	Default Rule DHCP	Any	Any	DHCP-Server DHCP-Client	ALLOW	Default Section Layer3	
5	Default Rule	Any	Any	Any	ALLOW	Default Section Layer3	

Figure 5.5 – Distributed firewall rules for source

In vRealize, we can analyze almost everything. For example, we need to know in the past day, what was the communication of Vlan 503 to whole networks. As we can see in Figure 5.6, we can track the communication of Vlan 503 to other networks.

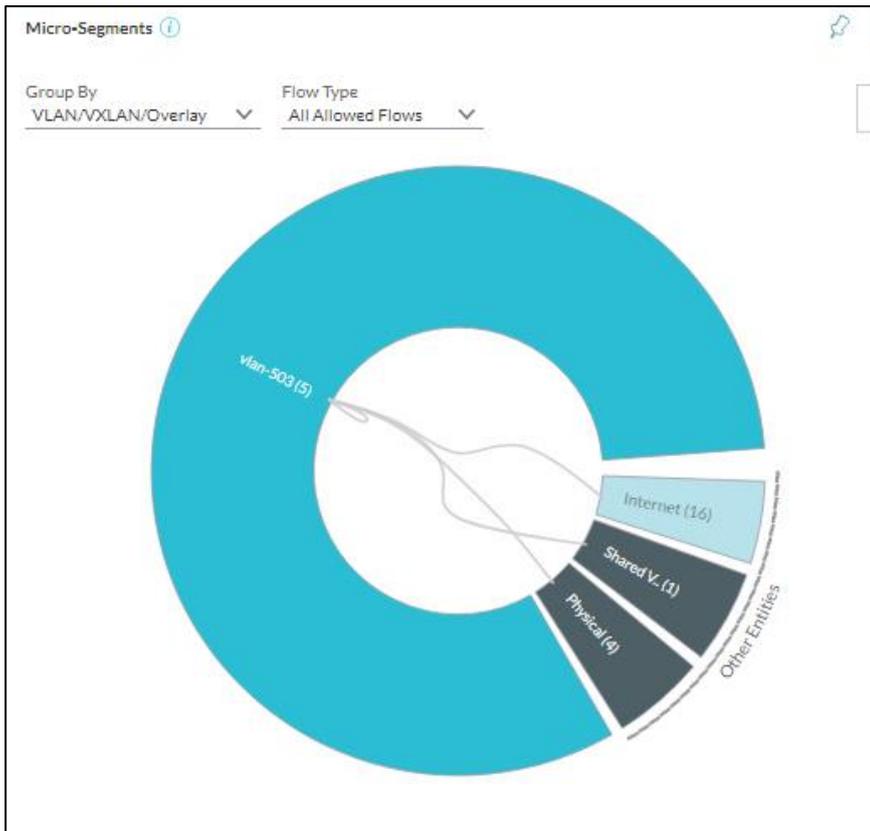


Figure 5.6

By clicking on each flow, we can see the list of flows and all firewall recommendations for this flow Figure 5.7

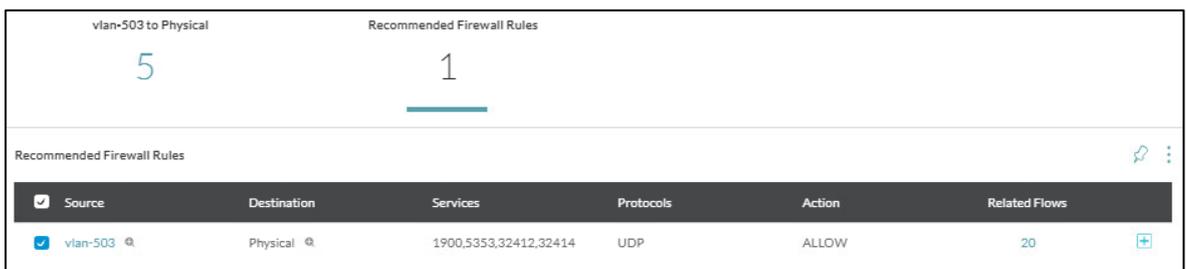


Figure 5.7

As we can see in figure 5.7, the recommended firewall rule generated to secure and segment traffic from the rest of the VLANs. One of the great features of the vRealize network is path tracking. It is beneficial for troubleshooting when analyzing the path of flows is difficult in NSX.

Suppose we want to track one of the shopping application traffic, which is forwarding to a database server. By filtering the specific source and destination by Their IP or DNS-name, we can reach a diagram 5.8.

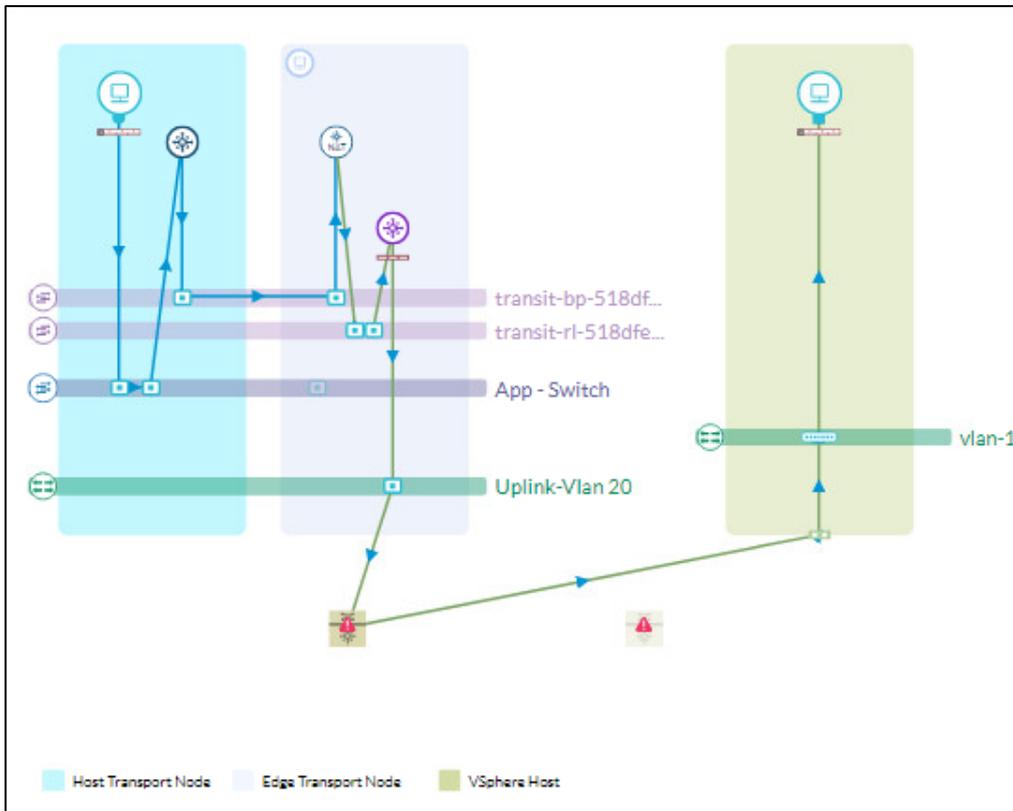


Figure 5.8

As we can see, not only we have an end to end view, but also, we can view in detail all the nodes configurations. As an example, by opening the source pc server, we can see all the configurations such as; IP, security group, NSX manager, gateway. Figure 5.9

Shopping-App				
IP Address 172.16.25.150	Disks Shopping-App-Hard disk 1	Datastore Unity350	Host w1-vmni-tmm-esx005.cmbu.local	Cluster South
Security Groups App	NSX-T Security Groups App	Version vmx-13	Datacenter Datacenter	Manager vc-south.vmi.cmbu.local
NSX Manager nsx-south.vmi.cmbu.local	Reserved CPU (MHz) 0	Reserved Memory (MB) 0	Nat Device LR-Tier-1	Power State On
Connection State Connected	CPU Cores 1	Memory (GB) 1	Firewall Status Protected	VNIC Count 1
Def Gateway 172.16.25.1	Default Gateway Router LR-Tier-1	Default Gateway Router Interfa... App-Port	Network Address 172.16.25.0/24	Disconnected VNIC Count 0
OS Ubuntu Linux (64-bit)	Resource Pool Mgmt	NSX-T Logical Port Shopping-App/Shopping-App.v...	FQDN shopping-app	

Figure 5.9

## 5.3 Application discovery

In an enterprise-level, when we have several applications or where there are multiple tiers in an application, creating an application by using public APIs or the user interface becomes time-consuming. vRealize network insight helps to automatically add the applications and their tiers and reduce a lot of manual efforts.

In the application discovery tab, we can see a list of all applications and their status, their protection, or applications with problems. Figure 5.10

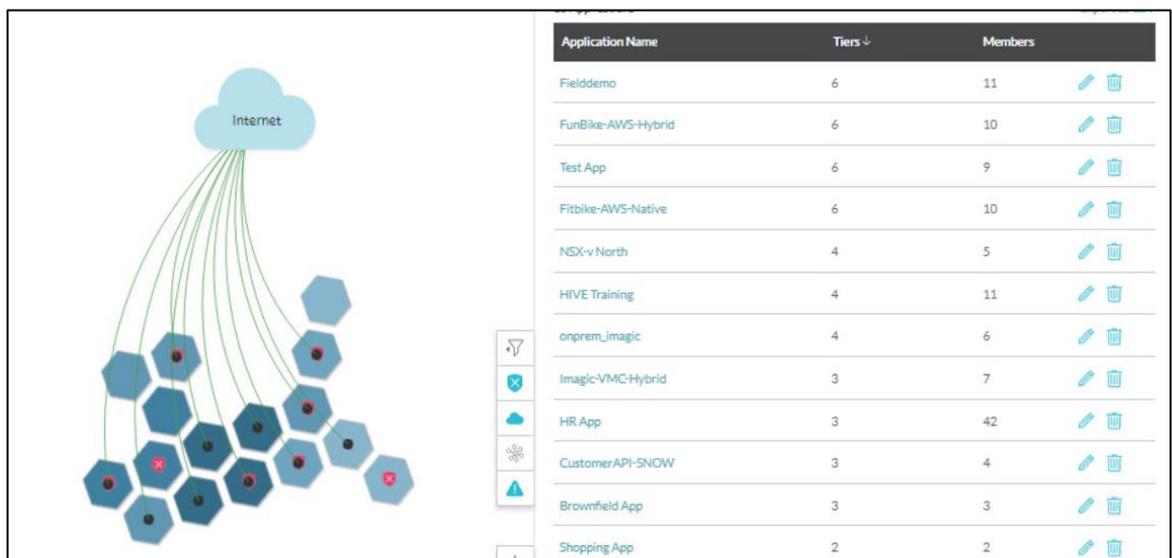


Figure 5.10 -application list with all status and protections

In this step, we need to create a new application automatically with all its tiers and connect it to the internet.

The form shows the creation of a new application with the following details:

- Application Name: Peiman Roufarshbaf application
- Tier/Deployment: (expanded)
- Name: Peiman Roufarshbaf
- Members: 3 VMs (VM Names: 'Shopping-App', '3TierApp01-DLR-0', '3TierApp01-Edge-0')

Figure 5.11 – New application creation in vRNI

We named this new application, Peiman Roufarshbaf, and we joined it to shopping applications and 3tierapp01, DLR-0, and edge0.

As can be seen in figure 5.12, the application is created and connected to the internet successfully.

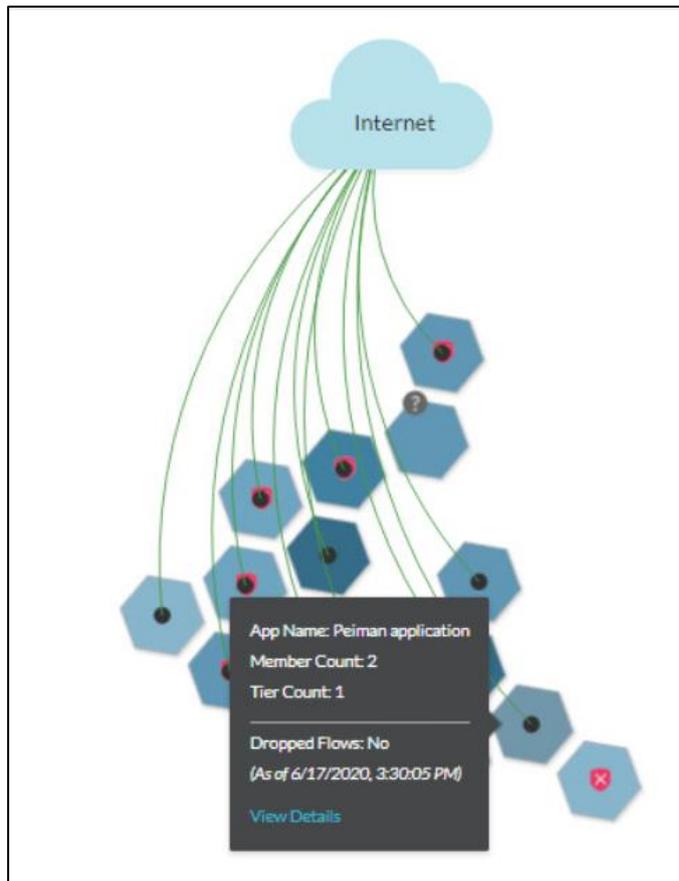


Figure 5.12

## Chapter 5

### Conclusion

“In this chapter, we used another useful product of VMware, which is vRealize Network Insight.

We used this product in the final chapter after all configuration and designed to bring intelligent operations for SDN and having a global view across both physical and virtual networks by collecting information from the virtual distributed switch and representing all this information through a graph. As time pass, the networks becoming more and more complex, and troubleshooting will be harder. vRealize insight gives us the ability to not only monitor the whole network from a central point but also track the flows generated from a source to the destination with all firewall recommendations. For example, if there is a misbehavior in the system or one part of the network has overloaded. By using vRealize insight, we can monitor all traffic passing during a specific period of time that we filtered and find the trouble. Another useful feature is that we can quickly modify or add a node with all its connections to the network

## Bibliography

*VMware NSX Data Center for vSphere*. (2019, 05 31). Retrieved from docs.vmware.com: <https://docs.vmware.com/en/VMware-NSX-Data-Center-for-vSphere/6.4/com.vmware.nsx.install.doc/GUID-B715387F-983D-4458-B9FB-AD49FCE03E04.html>

1. (2019, 05 31). Retrieved from docs.vmware.com: <https://docs.vmware.com/en/VMware-vSphere/6.7/com.vmware.vsphere.networking.doc/GUID-B15C6A13-797E-4BCB-B9D9-5CBC5A60C3A6.html>

Moenster, H. (n.d.). VMware vSphere 6 Fault Tolerance Architecture and Performance.

Peyo, T. (n.d.). <https://geek-university.com/vmware-esxi/vsphere-distributed-resource-scheduler-drs-requirements/>.

Rouse, M. (2015, May). <https://searchvmware.techtarget.com/definition/VMware-Platform-Services-Controller-PSC>.

VMware. (2018). *DATASHEETVMWARE vREALIZE | 1VMWARE vREALIZE NETWORK INSIGHT AND VMWARE NETWORK INSIGHT*.

Yaqub, N. (March 23, 2012). Comparison of Virtualization Performance: VMware and KVM. 18-20.

