Table of Contents

Lab Overview - HOL-1826-02-NET - VMware NSX-T with Kubernetes ......................... 2
  Lab Guidance ........................................................................................................... 3
Module 1 - Introduction to Kubernetes and NSX-T (15 minutes) ................................... 9
  In this section, we will learn about Kubernetes. ..................................................... 10
  Introduction to NSX-T .......................................................................................... 18
  Module 1 Conclusion ............................................................................................ 29
Module 2 - Kubernetes namespaces and NSX-T (45 minutes) ..................................... 30
  Lets take a look at the Lab Layout ........................................................................ 31
  K8s Namespaces .................................................................................................... 38
  Creating Pods in K8s ............................................................................................. 58
  Module 2 Conclusion ............................................................................................ 73
Module 3- Kubernetes Services and Ingress rules (30 minutes) .................................... 74
  K8s Services and east-west load balancing using OVS ........................................ 75
  K8s Ingress ............................................................................................................ 87
  Module 3 Conclusion ............................................................................................ 96
Module 4 - Security with NSX-T and Kubernetes (15 minutes) ................................... 97
  Microsegmentation in K8s with NSX (15 minutes) ............................................. 98
  Module 4 Conclusion ........................................................................................... 115
Lab Overview - HOL-1826-02-NET - VMware NSX-T with Kubernetes
Lab Guidance

Note: It may take more than 90 minutes to complete this lab. You should expect to only finish 2-3 of the modules during your time. The modules are independent of each other so you can start at the beginning of any module and proceed from there. You can use the Table of Contents to access any module of your choosing.

The Table of Contents can be accessed in the upper right-hand corner of the Lab Manual.

In this lab, we will explore what Kubernetes is and how to leverage NSX-T with Kubernetes to control and manage the virtual network of your containers.

Lab Module List:

1. **Module 1 - Introduction to Kubernetes and NSX-T** (15 minutes) High level overview with details about Kubernetes and NSX-T
2. **Module 2 - Kubernetes namespaces and NSX T** (45 minutes) Creating Kubernetes namespaces and seeing how this is done with NSX-T
3. **Module 3 - Service and Ingress rules** (20 minutes) (Basic/Advanced) Create and deploy a new POD and update ingress rules
4. **Module 4 - Security with NSX-T and Kubernetes** (15 minutes) Leveraging NSX-T to create microsegmentation within Kubernetes

Lab Captains:

- Hadar Freehling, Staff SE, US

This lab manual can be downloaded from the Hands-on Labs Document site found here:

http://docs.hol.vmware.com

This lab may be available in other languages. To set your language preference and have a localized manual deployed with your lab, you may utilize this document to help guide you through the process:

Location of the Main Console

1. The area in the RED box contains the Main Console. The Lab Manual is on the tab to the Right of the Main Console.
2. A particular lab may have additional consoles found on separate tabs in the upper left. You will be directed to open another specific console if needed.
3. Your lab starts with 90 minutes on the timer. The lab can not be saved. All your work must be done during the lab session. But you can click the **EXTEND** to increase your time. If you are at a VMware event, you can extend your lab time twice, for up to 30 minutes. Each click gives you an additional 15 minutes. Outside of VMware events, you can extend your lab time up to 9 hours and 30 minutes. Each click gives you an additional hour.

Alternate Methods of Keyboard Data Entry

During this module, you will input text into the Main Console. Besides directly typing it in, there are two very helpful methods of entering data which make it easier to enter complex data.
Click and Drag Lab Manual Content Into Console Active Window

You can also click and drag text and Command Line Interface (CLI) commands directly from the Lab Manual into the active window in the Main Console.

Accessing the Online International Keyboard

You can also use the Online International Keyboard found in the Main Console.

1. Click on the Keyboard Icon found on the Windows Quick Launch Task Bar.
Click once in active console window

In this example, you will use the Online Keyboard to enter the "@" sign used in email addresses. The "@" sign is Shift-2 on US keyboard layouts.

1. Click once in the active console window.
2. Click on the **Shift** key.

Click on the @ key

1. Click on the "@ key".

Notice the @ sign entered in the active console window.
When you first start your lab, you may notice a watermark on the desktop indicating that Windows is not activated.

One of the major benefits of virtualization is that virtual machines can be moved and run on any platform. The Hands-on Labs utilizes this benefit and we are able to run the labs out of multiple datacenters. However, these datacenters may not have identical processors, which triggers a Microsoft activation check through the Internet.

Rest assured, VMware and the Hands-on Labs are in full compliance with Microsoft licensing requirements. The lab that you are using is a self-contained pod and does not have full access to the Internet, which is required for Windows to verify the activation. Without full access to the Internet, this automated process fails and you see this watermark.

This cosmetic issue has no effect on your lab.

**Look at the lower right portion of the screen**
Please check to see that your lab is finished all the startup routines and is ready for you to start. If you see anything other than "Ready", please wait a few minutes. If after 5 minutes your lab has not changed to "Ready", please ask for assistance.
Module 1 - Introduction to Kubernetes and NSX-T (15 minutes)
In this section, we will learn about Kubernetes.

What is Kubernetes?

Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.

If you want to skip this section and start working on the lab click here.

Capabilities:

- Deploy your applications quickly and predictably
- Scale your applications on the fly
- Seamlessly roll out new features
- Optimize use of your hardware by using only the resources you need

Role: Kubernetes (K8s) sits in the Container as a Service (CaaS) or Container orchestration layer

Kubernetes Roots

- **Kubernetes**: Kubernetes or K8s in short is the ancient Greek word for Helmsmen
- **K8s roots**: Kubernetes was championed by Google and is now backed by major enterprise IT vendors and users (including VMware)
- **Borg** is Google’s internal task scheduling system. Borg served as the blueprint for Kubernetes, but the code base is different.
Kubernetes Master Components

Master Components:

1. **API server**: Target for all operations to the data model. External API clients like the K8s CLI client, the dashboard Web-Service, as well as all external and internal components interact with the API server by ‘watching’ and ‘setting’ resources.

2. **Scheduler**: Monitors Container (Pod) resources on the API Server, and assigns Worker Nodes to run the Pods based on filters.

3. **Controller Manager**: Embeds the core control loops shipped with Kubernetes. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API Server and makes changes attempting to move the current state towards the desired state.

Distributed Key-Value Store Components:

4. **Etcd**: Is used as the distributed key-value store of Kubernetes.

**Watching**: In etcd and Kubernetes everything is centered around ‘watching’ resources. Every resource can be watched in K8s on etcd through the API Server.
Kubernetes Node Components

1. **Kubelet**: The Kubelet agent on the Nodes is watching for ‘PodSpecs’ to determine what it is supposed to run. Kubelet Instructs Container runtimes to run containers through the container runtime API interface.

2. **Docker**: Is the most used container runtime in K8s. However K8s is ‘runtime agnostic’, and the goal is to support any runtime through a standard interface (CRI-O).

3. **Kube-Proxy**: Is a daemon watching the K8s ‘services’ on the API Server and implements east/west load-balancing on the nodes using NAT in IPTables.
Kubernetes Pod

1. **POD**: A pod (as in a pod of whales or pea pod) is a group of one or more containers
2. **Networking**: Containers within a pod share an IP address and port space, and can find each other via localhost. They can also communicate with each other using standard inter-process communications like SystemV semaphores or POSIX shared memory
3. **Pause Container**: A service container named ‘pause’ is created by Kubelet. Its sole purpose is to own the network stack (linux network namespace) and build the ‘low level network plumbing’

- **External Connectivity**: Only the pause container is started with an IP interface
- **Storage**: Containers in a Pod also share the same data volumes
- **Motivation**: Pods are a model of the pattern of multiple cooperating processes which form a cohesive unit of service

Kubernetes Replication Controller (rc) and Replica Set (rs)
Kubernetes RC & RS

• **Replication Controller:**

The replication controller enforces the 'desired' state of a collection of Pods. E.g. it makes sure that 4 Pods are always running in the cluster. If there are too many Pods, it will kill some. If there are too few, the Replication Controller will start more. Unlike manually created pods, the pods maintained by a Replication Controller are automatically replaced if they fail, get deleted, or are terminated.

• **Replica Set:**

Replica Set is the next-generation Replication Controller. It is in beta state right now. The only difference between a Replica Set and a Replication Controller right now is the selector support vs. Replication Controllers that only supports equality-based selector requirements.

Kubernetes Service

• **Gist:** A Kubernetes Service is an abstraction which defines a logical set of Pods

• **East/West Load-Balancing:** In terms of networking, a service usually contains a cluster IP, which is used as a Virtual IP reachable internally on all Nodes

• **IPTables:** In the default upstream implementation, IPTables is used to implement distributed east/west load-balancing

• **DNS:** A service is also represented with a DNS names, e.g. ‘redis-slave.cluster.local’ in the Kubernetes dynamic DNS service (SkyDNS) or through environment variable injection
• **External Access**: A K8s Service can also be made externally reachable through all Nodes IP interface using ‘NodePort’ exposing the Service through a specific UDP/TCP Port

• **Type**: In addition to ClusterIP and NodePort, some cloud providers like GCE support using the type ‘LoadBalancer’ to configure an external LoadBalancer to point to the Endpoints (Pods)

**CoreDNS (aka SkyDNS)**

**SkyDNS (Now called CoreDNS)** is a Service designed to provide service discovery

- It uses multiple central servers that are strongly-consistent and fault-tolerant
- Nodes register services using an HTTP API, and queries can be made over HTTP or DNS to perform discovery
- SkyDNS runs as K8s Pods (Containers) on the K8s Cluster
Kubernetes N/S Load-Balancing

• **N/S Load-Balancing**: Can be achieved using various solutions in K8s, this includes:
  - K8s Service of type ‘LoadBalancer’ which is watched by external logic to configure an external LoadBalancer
  - Statically configured external LoadBalancer (e.g. F5) that sends traffic to a K8s Service over ‘NodePort’ on specific Nodes
  - K8s Ingress; A K8s object that describes a N/S LoadBalancer. The K8s Ingress Object is ‘watched’ by a Ingress Controller that configures the LoadBalancer Datapath. Usually both the Ingress Controller and the LoadBalancer Datapath are running as a Pod

• **OpenShift ‘Router’**: In OpenShift a K8s Ingress ‘like’ LoadBalancer called ‘OpenShift Router’ is used. It is based on HA Proxy, alternatively an external F5 LB can be used
Kubernetes Namespace

- **Intent:** Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.
- **Name uniqueness:** Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces.
- **Quotas:** Namespaces are a way to divide cluster resources between multiple uses (via resource quota).
- **Multitenant Networking:** Network policy, a feature that is currently in Beta, will allow network segregation through Firewall Policies attached to Namespaces. OpenShift already supports ‘all or nothing’ network segregation between Namespaces (called Projects in OpenShift).
- **Authorization:** Also currently in Beta, per-Namespace access-control to Objects in specific namespaces will be made possible. OpenShift already supports this today.

```yaml
Namespace: foo
Base URI: /api/v1/namespaces/foo

redis-master' Pod:
/api/v1/namespaces/foo/pods/redis-master

redis' service:
/api/v1/namespaces/foo/services/redis-master

Namespace: bar
Base URI: /api/v1/namespaces/bar

redis-master' Pod:
/api/v1/namespaces/bar/pods/redis-master

redis' service:
/api/v1/namespaces/bar/services/redis-master
```
Introduction to NSX-T

In this module you will be introduced to NSX-T, its capabilities, and the components that make up the solution.

Lab Topology

What is NSX-T?

VMware NSX-T is designed to address emerging application frameworks and architectures that have heterogeneous endpoints and technology stacks. In addition to vSphere, these environments may also include other hypervisors (KVM), containers, and public clouds. NSX-T allows IT and development teams to choose the technologies best suited for their particular applications. NSX-T is also designed for management, operations and consumption by development organizations – in addition for IT.

In much the same way that server virtualization programmatically creates, snapshots, deletes and restores software-based virtual machines (VMs), NSX-T network virtualization programmatically creates, snapshots, deletes, and restores software-based virtual networks.
With network virtualization, the functional equivalent of a network hypervisor reproduces the complete set of Layer 2 through Layer 7 networking services (for example, switching, routing, access control, firewalling, QoS) in software. As a result, these services can be programmatically assembled in any arbitrary combination, to produce unique, isolated virtual networks in a matter of seconds.

NSX-T works by implementing three separate but integrated planes: management, control, and data. The three planes are implemented as a set of processes, modules, and agents residing on three types of nodes: manager, controller, and transport nodes.

- Every node hosts a management plane agent.
- The NSX Manager node hosts API services. Each NSX-T installation supports a single NSX Manager node and does not support an NSX Manager cluster.
- NSX Controller nodes host the central control plane cluster daemons.
- NSX Manager and NSX Controller nodes may be co-hosted on the same physical server.
- Transport nodes host local control plane daemons and forwarding engines.

**NSX-T Components (Part 1)**

![NSX architecture and components](image-url)

**Data Plane**
Performs stateless forwarding/transformation of packets based on tables populated by the control plane and reports topology information to the control plane, and maintains packet level statistics.

The data plane is the source of truth for the physical topology and status for example, VIF location, tunnel status, and so on. If you are dealing with moving packets from one place to another, you are in the data plane. The data plane also maintains status of and handles failover between multiple links/tunnels. Per-packet performance is paramount with very strict latency or jitter requirements. Data plane is not necessarily fully contained in kernel, drivers, userspace, or even specific userspace processes. Data plane is constrained to totally stateless forwarding based on tables/rules populated by control plane.

The data plane also may have components that maintain some amount of state for features such as TCP termination. This is different from the control plane managed state such as MAC:IP tunnel mappings, because the state managed by the control plane is about how to forward the packets, whereas state managed by the data plane is limited to how to manipulate payload.

**Control Plane**

Computes all ephemeral runtime state based on configuration from the management plane, disseminates topology information reported by the data plane elements, and pushes stateless configuration to forwarding engines.

The control plane is sometimes described as the signaling for the network. If you are dealing with processing messages in order to maintain the data plane in the presence of static user configuration, you are in the control plane (for example, responding to a vMotion of a virtual machine (VM) is a control plane responsibility, but connecting the VM to the logical network is a management plane responsibility) Often the control plane is acting as a reflector for topological info from the data plane elements to one another for example, MAC/Tunnel mappings for TEPs. In other cases, the control plane is acting on data received from some data plane elements to (re)configure some data plane elements such as, using VIF locators to compute and establish the correct subset mesh of tunnels.

The set of objects that the control plane deals with include VIFs, logical networks, logical ports, logical routers, IP addresses, and so on.

The control plane is split into two parts in NSX-T, the central control plane (CCP), which runs on the NSX Controller cluster nodes, and the local control plane (LCP), which runs on the transport nodes, adjacent to the data plane it controls. The Central Control Plane computes some ephemeral runtime state based on configuration from the management plane and disseminates information reported by the data plane elements via the local control plane. The Local Control Plane monitors local link status, computes most ephemeral runtime state based on updates from data plane and CCP, and pushes
stateless configuration to forwarding engines. The LCP shares fate with the data plane element which hosts it.

**Management Plane**

The management plane provides a single API entry point to the system, persists user configuration, handles user queries, and performs operational tasks on all management, control, and data plane nodes in the system.

For NSX-T anything dealing with querying, modifying, and persisting user configuration is a management plane responsibility, while dissemination of that configuration down to the correct subset of data plane elements is a control plane responsibility. This means that some data belongs to multiple planes depending on what stage of its existence it is in. The management plane also handles querying recent status and statistics from the control plane, and sometimes directly from the data plane.

The management plane is the one and only source-of-truth for the configured (logical) system, as managed by the user via configuration. Changes are made using either a RESTful API or the NSX-T UI.

In NSX there is also a management plane agent (MPA) running on all cluster and transport nodes. Example use cases are bootstrapping configurations such as central management node address(es) credentials, packages, statistics, and status. The MPA can run relatively independently of the control plane and data plane, and to be restarted independently if its process crashes or wedges, however, there are scenarios where fate is shared because they run on the same host. The MPA is both locally accessible and remotely accessible. MPA runs on transport nodes, control nodes, and management nodes for node management. On transport nodes it may perform data plane related tasks as well.

Tasks that happen on the management plan include:

- Configuration persistence (desired logical state)
- Input validation
- User management - role assignments
- Policy management
- Background task tracking

**NSX-T Components (Part 2)**

**NSX Manager**

NSX Manager provides the graphical user interface (GUI) and the REST APIs for creating, configuring, and monitoring NSX-T components, such as controllers, logical switches, and edge services gateways.
NSX Manager is the management plane for the NSX-T eco-system. NSX Manager provides an aggregated system view and is the centralized network management component of NSX-T. It provides a method for monitoring and troubleshooting workloads attached to virtual networks created by NSX-T. It provides configuration and orchestration of:

- Logical networking components – logical switching and routing
- Networking and Edge services
- Security services and distributed firewall - Edge services and security services can be provided by either built-in components of NSX Manager or by integrated 3rd party vendors.

NSX Manager allows seamless orchestration of both built-in and external services. All security services, whether built-in or 3rd party, are deployed and configured by the NSX-T management plane. The management plane provides a single window for viewing services availability. It also facilitates policy based service chaining, context sharing, and inter-service events handling. This simplifies the auditing of the security posture, streamlining application of identity-based controls (for example, AD and mobility profiles).

NSX Manager also provides REST API entry-points to automate consumption. This flexible architecture allows for automation of all configuration and monitoring aspects via any cloud management platform, security vendor platform, or automation framework.

The NSX-T Management Plane Agent (MPA) is an NSX Manager component that lives on each and every node (hypervisor). The MPA is in charge of persisting the desired state of the system and for communicating non-flow-controlling (NFC) messages such as configuration, statistics, status and real time data between transport nodes and the management plane.

**NSX Controller**

NSX Controller is an advanced distributed state management system that controls virtual networks and overlay transport tunnels.

NSX Controller is deployed as a cluster of highly available virtual appliances that are responsible for the programmatic deployment of virtual networks across the entire NSX-T architecture. The NSX-T Central Control Plane (CCP) is logically separated from all data plane traffic, meaning any failure in the control plane does not affect existing data plane operations. Traffic doesn’t pass through the controller; instead the controller is responsible for providing configuration to other NSX Controller components such as the logical switches, logical routers, and edge configuration. Stability and reliability of data transport are central concerns in networking. To further enhance high availability and scalability, the NSX Controller is deployed in a cluster of three instances.
**Logical Switches**

The logical switching capability in the NSX Edge platform provides the ability to spin up isolated logical L2 networks with the same flexibility and agility that exists for virtual machines.

A cloud deployment for a virtual data center has a variety of applications across multiple tenants. These applications and tenants require isolation from each other for security, fault isolation, and to avoid overlapping IP addressing issues. Endpoints, both virtual and physical, can connect to logical segments and establish connectivity independently from their physical location in the data center network. This is enabled through the decoupling of network infrastructure from logical network (i.e., underlay network from overlay network) provided by NSX-T network virtualization.

A logical switch provides a representation of Layer 2 switched connectivity across many hosts with Layer 3 IP reachability between them. If you plan to restrict some logical networks to a limited set of hosts or you have custom connectivity requirements, you may find it necessary to create additional logical switch.

**NSX-T Components (Part 3)**

**Logical Routers**

NSX-T logical routers provide North-South connectivity, thereby enabling tenants to access public networks, and East-West connectivity between different networks within the same tenants.

A logical router is a configured partition of a traditional network hardware router. It replicates the hardware's functionality, creating multiple routing domains within a single router. Logical routers perform a subset of the tasks that can be handled by the physical router, and each can contain multiple routing instances and routing tables. Using logical routers can be an effective way to maximize router usage, because a set of logical routers within a single physical router can perform the operations previously performed by several pieces of equipment.

With NSX-T it’s possible to create two-tier logical router topology: the top-tier logical router is Tier 0 and the bottom-tier logical router is Tier 1. This structure gives both provider administrator and tenant administrators complete control over their services and policies. Administrators control and configure Tier-0 routing and services, and tenant administrators control and configure Tier-1. The north end of Tier-0 interfaces with the physical network, and is where dynamic routing protocols can be configured to exchange routing information with physical routers. The south end of Tier-0 connects to multiple Tier-1 routing layer(s) and receives routing information from them. To optimize resource usage, the Tier-0 layer does not push all the routes coming from the physical network towards Tier-1, but does provide default information.
Southbound, the Tier-1 routing layer interfaces with the logical switches defined by the tenant administrators, and provides one-hop routing function between them. For Tier-1 attached subnets to be reachable from the physical network, route redistribution towards Tier-0 layer must be enabled. However, there isn’t a classical routing protocol (such as OSPF or BGP) running between Tier-1 layer and Tier-0 layer, and all the routes go through the NSX-T control plane. Note that the two-tier routing topology is not mandatory, if there is no need to separate provider and tenant, a single tier topology can be created and in this scenario the logical switches are connected directly to the Tier-0 layer and there is no Tier-1 layer.

A logical router consists of two optional parts: a distributed router (DR) and one or more service routers (SR).

A DR spans hypervisors whose VMs are connected to this logical router, as well as edge nodes the logical router is bound to. Functionally, the DR is responsible for one-hop distributed routing between logical switches and/or logical routers connected to this logical router. The SR is responsible for delivering services that are not currently implemented in a distributed fashion, such as stateful NAT.

A logical router always has a DR, and it has SRs if any of the following is true:

- The logical router is a Tier-0 router, even if no stateful services are configured
- The logical router is Tier-1 router linked to a Tier-0 router and has services configured that do not have a distributed implementation (such as NAT, LB, DHCP)

The NSX-T management plane (MP) is responsible for automatically creating the structure that connects the service router to the distributed router. The MP creates a transit logical switch and allocates it a VNI, then creates a port on each SR and DR, connects them to the transit logical switch, and allocates IP addresses for the SR and DR.

**NSX Edge Node**

NSX Edge Node provides routing services and connectivity to networks that are external to the NSX-T deployment.

With NSX Edge Node, virtual machines or workloads that reside on the same host on different subnets can communicate with one another without having to traverse a traditional routing interface.

NSX Edge Node is required for establishing external connectivity from the NSX-T domain, through a Tier-0 router via BGP or static routing. Additionally, an NSX Edge Node must be deployed if you require network address translation (NAT) services at either the Tier-0 or Tier-1 logical routers.
The NSX Edge Node connects isolated, stub networks to shared (uplink) networks by providing common gateway services such as NAT, and dynamic routing. Common deployments of NSX Edge Node include in the DMZ and multi-tenant Cloud environments where the NSX Edge Node creates virtual boundaries for each tenant.

Transport Zones

A transport zone controls which hosts a logical switch can reach. It can span one or more host clusters. Transport zones dictate which hosts and, therefore, which VMs can participate in the use of a particular network.

A Transport Zone defines a collection of hosts that can communicate with each other across a physical network infrastructure. This communication happens over one or more interfaces defined as Tunnel Endpoints (TEPs).

If two transport nodes are in the same transport zone, VMs hosted on those transport nodes can "see" and therefore be attached to NSX-T logical switches that are also in that transport zone. This attachment makes it possible for the VMs to communicate with each other, assuming that the VMs have Layer 2/Layer 3 reachability. If VMs are attached to switches that are in different transport zones, the VMs cannot communicate with each other. Transport zones do not replace Layer 2/Layer 3 reachability requirements, but they place a limit on reachability. Put another way, belonging to the same transport zone is a prerequisite for connectivity. After that prerequisite is met, reachability is possible but not automatic. To achieve actual reachability, Layer 2 and (for different subnets) Layer 3 networking must be operational.

A node can serve as a transport node if it contains at least one hostswitch. When you create a host transport node and then add the node to a transport zone, NSX-T installs a hostswitch on the host. For each transport zone that the host belongs to, a separate hostswitch is installed. The hostswitch is used for attaching VMs to NSX-T logical switches and for creating NSX-T logical router uplinks and downlinks.

Glossary of Components

The common NSX-T concepts that are used in the documentation and user interface.

Control Plane

Computes runtime state based on configuration from the management plane. Control plane disseminates topology information reported by the data plane elements, and pushes stateless configuration to forwarding engines.

Data Plane
Performs stateless forwarding or transformation of packets based on tables populated by the control plane. Data plane reports topology information to the control plane and maintains packet level statistics.

**External Network**

A physical network or VLAN not managed by NSX-T. You can link your logical network or overlay network to an external network through an NSX Edge. For example, a physical network in a customer data center or a VLAN in a physical environment.

**Fabric Node**

Node that has been registered with the NSX-T management plane and has NSX-T modules installed. For a hypervisor host or NSX Edge to be part of the NSX-T overlay, it must be added to the NSX-T fabric.

**Fabric Profile**

Represents a specific configuration that can be associated with an NSX Edge cluster. For example, the fabric profile might contain the tunneling properties for dead peer detection.

**Logical Router**

NSX-T routing entity.

**Logical Router Port**

Logical network port to which you can attach a logical switch port or an uplink port to a physical network.

**Logical Switch**

API entity that provides virtual Layer 2 switching for VM interfaces and Gateway interfaces. A logical switch gives tenant network administrators the logical equivalent of a physical Layer 2 switch, allowing them to connect a set of VMs to a common broadcast domain. A logical switch is a logical entity independent of the physical hypervisor infrastructure and spans many hypervisors, connecting VMs regardless of their physical location. This allows VMs to migrate without requiring reconfiguration by the tenant network administrator.

In a multi-tenant cloud, many logical switches might exist side-by-side on the same hypervisor hardware, with each Layer 2 segment isolated from the others. Logical switches can be connected using logical routers, and logical routers can provide uplink ports connected to the external physical network.

**Logical Switch Port**
Logical switch attachment point to establish a connection to a virtual machine network interface or a logical router interface. The logical switch port reports applied switching profile, port state, and link status.

Management Plane

Provides single API entry point to the system, persists user configuration, handles user queries, and performs operational tasks on all of the management, control, and data plane nodes in the system. Management plane is also responsible for querying, modifying, and persisting use configuration.

NSX Controller Cluster

Deployed as a cluster of highly available virtual appliances that are responsible for the programmatic deployment of virtual networks across the entire NSX-T architecture.

NSX Edge Cluster

Collection of NSX Edge node appliances that have the same settings as protocols involved in high-availability monitoring.

NSX Edge Node

Component with the functional goal is to provide computational power to deliver the IP routing and the IP services functions.

NSX-T Hostswitch or KVM Open vSwitch

Software that runs on the hypervisor and provides physical traffic forwarding. The hostswitch or OVS is invisible to the tenant network administrator and provides the underlying forwarding service that each logical switch relies on. To achieve network virtualization, a network controller must configure the hypervisor hostswitches with network flow tables that form the logical broadcast domains the tenant administrators defined when they created and configured their logical switches.

Each logical broadcast domain is implemented by tunneling VM-to-VM traffic and VM-to-logical router traffic using the tunnel encapsulation mechanism Geneve. The network controller has the global view of the data center and ensures that the hypervisor hostswitch flow tables are updated as VMs are created, moved, or removed.

NSX Manager

Node that hosts the API services, the management plane, and the agent services.

Open vSwitch (OVS)

Open source software switch that acts as a hypervisor hostswitch within XenServer, Xen, KVM, and other Linux-based hypervisors. NSX Edge switching components are based on OVS.
**Overlay Logical Network**

Logical network implemented using Layer 2-in-Layer 3 tunneling such that the topology seen by VMs is decoupled from that of the physical network.

**Physical Interface (pNIC)**

Network interface on a physical server that a hypervisor is installed on.

**Tier-0 Logical Router**

Provider logical router is also known as Tier-0 logical router interfaces with the physical network. Tier-0 logical router is a top-tier router and can be realized as active-active or active-standby cluster of services router. The logical router runs BGP and peers with physical routers. In active-standby mode the logical router can also provide stateful services.

**Tier-1 Logical Router**

Tier-1 logical router is the second tier router that connects to one Tier-0 logical router for northbound connectivity and one or more overlay networks for southbound connectivity. Tier-1 logical router can be an active-standby cluster of services router providing stateful services.

**Transport Zone**

Collection of transport nodes that defines the maximum span for logical switches. A transport zone represents a set of similarly provisioned hypervisors and the logical switches that connect VMs on those hypervisors. NSX-T can deploy the required supporting software packages to the hosts because it knows what features are enabled on the logical switches.

**VM Interface (vNIC)**

Network interface on a virtual machine that provides connectivity between the virtual guest operating system and the standard vSwitch or vSphere distributed switch. The vNIC can be attached to a logical port. You can identify a vNIC based on its Unique ID (UUID).

**TEP**

Tunnel End Point. Tunnel endpoints enable hypervisor hosts to participate in an NSX-T overlay. The NSX-T overlay deploys a Layer 2 network on top of an existing Layer 3 network fabric by encapsulating frames inside of packets and transferring the packets over an underlying transport network. The underlying transport network can be another Layer 2 networks or it can cross Layer 3 boundaries. The TEP is the connection point at which the encapsulation and decapsulation takes place.
Module 1 Conclusion

In this Module you learned about Kubernetes and NSX-T. Now we can use what you learned in the next module.

You've finished Module 1

Congratulations on completing Module 1.

Proceed to any module below which interests you most.

- **Module 2 - Kubernetes namespaces and NSX T** (45 minutes) Creating Kubernetes namespaces and seeing how this is done with NSX-T
- **Module 3 - Service and Ingress rules** (20 minutes) (Basic/Advanced) Create and deploy a new POD and update ingress rules
- **Module 4 - Security with NSX-T and Kubernetes** (15 minutes) Leveraging NSX-T to create microsegmentation within Kubernetes

How to End Lab

To end your lab click on the **END** button.
Module 2 - Kubernetes namespaces and NSX-T (45 minutes)
Let's take a look at the Lab Layout

The Kubernetes environment of this lab is located behind a NSX Tier 0 (provider) virtual router. Review the IP space that will be used for this lab in the attached picture.

vCenter view of the lab
For Reference only:

Here is the lab layout as seen through vCenter.

1. The 'RegionA01-MGMT' cluster runs the NSX-T control plane and gateway only. Hypervisors are not prepared as transport nodes
2. The 'RegionA01-COMP01' cluster runs the Kubernetes Node VMs, and its Hypervisors are prepared as transport nodes

The Kubernetes Node VMs are connected to two logical switches,

3. 'k8s-mgmt' is the logical switch for the mgmt connections to the Node.
4. 'k8s-node-vifs' is the logical switch for the Pod traffic.

**Check Lab status**

1. Click on Putty icon on the task bar
Access the K8 Master

1. Select K8Master from the list
2. Click on Load
3. Click on Open

Log into K8 Node
Now you are in.

**Status Check**

Check the status of the nodes by typing `kubectl get nodes`.

```
kubectl get nodes
```

Status should return as Ready.
Check the SkyDNS status

1. You need to check that the SkyDNS is running. To check status:

   ```
kubectl -n kube-system get pods
   ```

   Make sure that the kube-dns- is Running. If not, perform the following steps.

Kube-DNS in CrashLoopBackOff state.
If the kube-dns is not in Running status you will need to delete the DNS service and restart it by performing the following steps.

**Delete kube-dns service.**

1. Delete the kube-dns-xxxx service. Replace the xxxxxx with the actual instance number found in your pod.

   ```bash
   kubectl delete -n kube-system po/kube-dns-xxxxxx
   ```

   **NOTE:** You can highlight the number and then left click. This will paste what you have highlighted to the line you are typing.

   For the above example it would be:
   
   "kubectl delete -n kube-system po/kube-dns-3913472980-f4kw9"

**Confirm kube-dns service has restarted**

```
localadmin@k8s-master:~$ kubectl -n kube-system get pods
NAME                    READY STATUS    RESTARTS AGE
etcd-k8s-master         1/1 Running    6          28d
kube-apiserver-k8s-master 1/1 Running    6          28d
kube-controller-manager-k8s-master 1/1 Running    6          28d
kube-dns-3913472980-v0j2v 2/3 Running    0          17s
kube-proxy-2m17r         1/1 Running    6          28d
kube-proxy-63gyp         1/1 Running    5          28d
kube-proxy-zf8bl         1/1 Running    7          28d
kube-scheduler-k8s-master 1/1 Running    6          28d
```
1. Reissue the `get pods` command to verify the kube-dns service is running. You can use the up arrow to replay the command or use the command below.

```bash
kubectl -n kube-system get pods
```

You will see that the service has restarted with a new unique ID and is Running.
**K8s Namespaces**

By default, all resources in Kubernetes cluster are created in a default namespace. A pod will run with unbounded CPU and memory requests/limits. A kubernetes namespace allows to partition created resources into a logically named group. Each namespace provides:

- **a unique scope** for resources to avoid name collisions
- **policies** to ensure appropriate authority to trusted users
- ability to specify **constraints for resource consumption**

This allows a Kubernetes cluster to share resources by multiple groups and provide different levels of QoS for each group. Resources created in one namespace are hidden from other namespaces. Multiple namespaces can be created, each potentially with different constraints.

**Namespaces and NSX**

Adding NSX to Kubernetes provides us a whole new set of capabilities.
• **Namespaces**: We are dynamically building a separate network topology per K8s namespace, every K8s namespace gets one or more logical switches and one Tier-1 router.

• **Nodes**: Are not doing IP routing, every Pod has its own logical port on a NSX logical switch. Every Node can have Pods from different Namespaces and with this from different IP Subnets / Topologies

• **Firewall**: Every Pod has DFW rules applied on its Interface

• **Routing**: High performant East/West and North/South routing using NSX’s routing infrastructure, including dynamic routing to physical network

• **Visibility and troubleshooting**: Every Pod has a logical port on the logical switch with:
  - Counters
  - SPAN / Remote mirroring
  - IPFIX export
  - Traceflow / Port-Connection tool
  - Spoofguard

• **IPAM**: NSX is used to provide IP Address Management by supplying Subnets from IP Block to Namespaces, and Individual IPs and MAC to Pods

### Network Container Plugin (NCP)
1. **NSX Container Plugin:** NCP is a software component provided by VMware in form of a container image, e.g. to be run as a K8s Pod
2. **Adapter layer:** NCP is built in a modular way, so that individual adapters can be added for different CaaS and PaaS systems
3. **NSX Infra layer:** Implements the logic that creates topologies, attaches logical ports, etc. based on triggers from the Adapter layer
4. **NSX API Client:** Implements a standardized interface to the NSX API

**NCP Workflow**

**Namespace Creation workflow**

1. **NCP** creates a watch (a service that is looking for something else to happen) on K8s API for any Namespace events
2. A user creates a new K8s Namespace
3. The K8s API Server notifies NCP of the change (addition) of Namespaces
4. **NCP** creates the network topology for the Namespace
   - a) Requests a new subnet from the pre-configured IP block in NSX
   - b) Creates a logical switch
c) Creates a T1 router and attaches it to the pre-configured global T0 router

- Creates a router port on the T1 router, attaches it to the LS, and assigns an IP from the new subnet

**Existing namespaces**

Let's see what we have for existing namespaces. If you kept your Putty session open to K8Master, please proceed to the next step, otherwise perform these tasks.

1. Click on **Putty icon** on the task bar

   1. Select **K8Master** from the list
   2. Click on **Load**
   3. Click on **Open**
Existing namespaces cont...

1. Let's see what namespaces already exist. Type the follow:

```
kubectl get ns
```

You should see three different namespaces: default, kube-public, and kube-system

Kubernetes namespaces and NSX

Now that we have seen the namespaces in Kubernetes, we will look for those namespaces in NSX. Since a namespace is equivalent to a network switch, we should see Kubernetes equivalent switches within NSX.

Click on Google Chrome

Launch NSX Manager

1. Click on the nsxmgr-01a link in the toolbar
Login to NSX

1. For the User Name, enter admin
2. For Password, enter: VMware1!
3. Click on Log In
Within the NSX manager, you will see a host of features and capabilities that you can access. For now, we want to look at the switching function of NSX.

1. **Click on Switching** link
Here are all the logical switches that have been created within NSX. Some of these switches are being used for other parts of the lab and also to provide you preconfigured functionality. For the Kubernetes lab, we want to look at the switches that correlate to our namespaces.

Look at the list of switches you will find the three switches that are the same name as the Kubernetes namespace you saw on the K8 Master. These were created by NCP.

The switches are:

- k8s-cl1-default-0
- k8s-cl1-kube-public -0
- k8s-cl1-kube-system -0

1. **Click on the k8s-cl1-kube-public-0**
NSX switch info

Clicking on the switch name brings up the Summary page and more information about the switch.

1. If it isn't open, **click on the Tags arrow** to open up the Tags information section. Tags are used by the Kubernetes NSX CNI to allow for their integration. For example, the cluster name in tags matches the cluster name in Kubernetes.
2. Click on the Subnets arrow. Note the subnet that is assigned to the switch. You will see how this aligns to the IP Pool integration with NSX later.

Repeat the steps on the other K8s switches to see what subnets they have assigned.
Now let's examine the routing that is setup for this lab and Kubernetes.

1. **Click on the Routing link** on the left hand side.
NSX Routers

There are several routers that have been setup both for Kubernetes and for the lab to function.

- The Tier0 router is our border router with the external network, while the Tier 1 type routers are for separating of routers per tenant of domain.
- For k8s, you will find the routers have matched our namespaces.
### Router ports

<table>
<thead>
<tr>
<th>DATE</th>
<th>TIME</th>
<th>USER</th>
<th>ACTION</th>
<th>OBJECT</th>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOL-1826-02-NET</td>
<td>11:00</td>
<td>Administrator</td>
<td>Added</td>
<td>k8s-cl1-lr-kube-system</td>
<td>Logical Router link</td>
</tr>
</tbody>
</table>

1. **Click on the k8s-cl1-lr-kube-system Logical Router link**
K8s NSX router info

Clicking on the router will bring up the Summary page along with other tabs. Take a moment to review the information provided on this page.

Router Ports
1. Click on the Configuration tab
2. Click on Router Ports

**Router Ports info**

Here you will find the ports configured on the Router along with the type of port and the IP address of the port. Take a moment to review the information.
NSX IP pools

Now we will look at the IP pools that are being used by K8s in this lab.

1. **Click on the DDI link** on the left hand
**IPAM Blocks**

1. Click on the IPAM link

**IPAM Blocks review**
Here you will find the K8s IP block that has been setup for this environment. NSX acts as the IPAM and provides IP addressing for Kubernetes. IP subnets for Namespaces are dynamically created out from a user defined IP Block.

1. **Click on the k8s-ip-block** link to review assigned Subnets for the IP Block
2. Click on the **Subnets** tab

**IP Block summary**

![IP Block summary table](image)

Here you find the IP block information in more details.

1. Now you can see the details about the subnets and their assignments. Here you see the different CIDR and IP Ranges assigned

**Creating a new namespace**

We will now create a new namespace within K8s and see what happens within NSX. Minimize your NSX session, you will need it in a moment.

1. If you have closed putty, **Click on Putty** icon on the task bar
Putty to K8 Master

1. Select **K8Master** from the list
2. Click on **Load**
3. Click on **Open**

Creating Namespace

1. To create a new namespace type the following

   ```bash
   kubectl create ns nsx-ujo
   ```

   This will return "namespace "nsx-ujo" created"
NSX Namespace check

Now, go back to your browser that is open to NSX Manager.

1. **Click on the Switching link** on the left side.

**Ujo switch**

Looking through the list you will now find your name switch = namespace of ujo. (If the switch does not appear, click the refresh icon at the bottom of the page or refresh the browser)
1. **Click on the switch** and review the subnet it was assigned.

Remembering what steps you took previously, check the routing and IP Pool information and see what was created, as related to the ujo namespace.

### Updated Lab Layout

After creating the nsx-ujo name space, the logical layout of the lab has been updated. Here is a visual of the new layout.
Creating Pods in K8s

CNI - Container Network Interface

CNI stands for Container Networking Interface and it’s goal is to create a generic plugin-based networking solution for containers. Container Network interfaces are used by Kubernetes, OpenShift, Cloud Foundry and Mesos

Implementation:

- Uses a simple local executable call and environment variable injection as the API between the container runtime / orchestration and the network plug

Relation to Docker Networking:

- The CNI spec is different to Dockers Libnetwork Container Network Model (CNM), that uses a HTTP REST API for the plugins and is more focused towards the ‘Docker CaaS Model’
NSX-T Container Interface (CIF)

Container Interfaces in NSX-T

K8s Node VMs: Most customers are looking to deploy K8s Nodes as VMs today

- Nested Network-Virtualization: Instead of terminating the overlay tunnels in the Node VM, we are extending the Hypervisor vSwitch into the Node VM using VLAN tagging. The Node VM vSwitch (OVS) is ‘standalone’, and only gets programmed by the NSX CNI Plugin.

Benefits:

- Enhanced security through strong isolation of the Node VM from the NSX Control-Plane
- Less transport-nodes in NSX which equates to higher scale
Pod Attachment Workflow

1. NCP creates a watch on K8s API for any Pod events
2. A user creates a new K8s Pod
3. The K8s API Server notifies NCP of the change (addition) of Pods
4. NCP creates a logical port:
   - Requests an IP from the Namespace/LS Subnet
   - Request a MAC from the container MAC pool in NSX
   - Assigns a VLAN for the Pod
   - Creates a logical port (Sub-VIF / CIF) on the Namespace LS and assigns the IP, MAC and VLAN to the logical port
   - Adds all K8s Pod Labels as Tags to the logical port
1. Once the logical port is created, the NSX LCP will create the LP on the hypervisor
2. A new NSX Service, the Hyperbus monitors LCP for new CIF interfaces and learns the CIF’s Id/IP/MAC/VLAN binding
3. Kubelet sees a new ‘PodSpec’ from the K8s Master and starts a new Pod. It executes the NSX cni plugin binary to do the ‘network wiring’ of the Pod – This call is proxy'd to the NSX Node Agent
4. The NSX Node Agent gets the CIF’s Id/IP/MAC/VLAN binding data from the Hyperbus over the isolated and secured channel (one-way connection establishment)
5. The Node Agent creates the OVS port with the right VLAN, and configures the Pods network interface to connect to OVS with the received IP/MAC. After this, Kubelet is un-blocked and the Pod creation succeeds.

Creating a POD

1. If you have closed putty, **Click on Putty icon** on the task bar

Putty to K8 Master

1. **Select K8Master** from the list
2. **Click on Load**
3. **Click on Open**
Switching default namespace

1. We now need to switch the default namespace to the one we created nsx-ujo. To do this, enter the following commands.

```
kubectl config use-context kubernetes-admin@kubernetes
```

This will return a "Switched "kubernetes-admin@kubernetes"

Next, type the following

```
kubectl config set-context kubernetes-admin@kubernetes --namespace nsx-ujo
```

This will return a "Context "kubernetes-admin@kubernetes""."
Review the yaml file

1. We have a created yaml file we will use to create a replication controller. First we will take a look at the file. Type the following two lines.

```
cd demo
vimcat nsx-demo-controller-no-labels.yaml
```

2. Type the second command.

```
vimcat nsx-demo-controller-no-labels.yaml
```

*vimcat* is used to help show the file in different colors.

Start the replication controller

```
kubectl create -f nsx-demo-controller-no-labels.yaml
replicationcontroller "nsx-demo-rc" created
```

Now to start the replication controller and watch the Pod being created.
1. Type the following:

```
kubectl create -f nsx-demo-controller-no-labels.yaml
```

You will receive a "replication controller "nsx-demo-rc" created"

**Watch the create**

```
kubectl get all -o wide
```

1. Now lets see what is occurring. Type the following:

```
kubectl get all -o wide
```

You will see a screen showing the creation of the Pods and the IP addresses assigned. Notice that each nsx-demo-rc- have a different name. Take note of the IPs assigned. We will use this info later.

**Check connectivity**

Now that the Pods are running, lets check connectivity. We will perform some ping tests. Pick any Pod and use it for the below. In this example, I use the last Pod, *nsx-demo-rc-wznfk* as the starting point for the pings.

Look at your list of *nsx-demo-xxxx* Pods and pick one of the names to use for the below commands. **The POD names are dynamically named**

1. To test between the Pods, type the following.

```
kubectl exec -it nsx-demo-rc-qs922 -- ping -c 2 10.4.0.37
```

2. Now, ping the router interface on the Tier0 router by typing the following command:

```
kubectl exec -it nsx-demo-rc-qs922 -- ping -c 2 10.0.1.1
```
Review the yaml file again

```yaml
localadmin@k8s-master:~$ cd demo
dllocaladmin@k8s-master:/demo$ vimcat nsx-demo-controller-no-labels.yaml

apiVersion: v1
kind: ReplicationController
metadata:
  name: nsx-demo-rc
  labels:
    app: nsx-demo
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nsx-demo
    spec:
      containers:
      - name: nsx-demo
        image: yfauser/nsx-demo
        ports:
          - containerPort: 80

localadmin@k8s-master:~$ cd demo
```

If you are still in the same demo directory, ignore the first line.

1. **Type the commands below.** Take note of the app label.

```bash
cd demo
vimcat nsx-demo-controller-no-labels.yaml
```
Review ports in NSX

If you have closed it, reopen Google Chrome and connect to NSX.

1. Once connected, **click on the Switching link** on the left hand side.
NSX-ujo switch

1. Click on the k8s-cl1-nsx-ujo switch
2. Click on the Related Tab
3. Click on Ports
Related Ports

If you look at the far right column, you can see that each logical port was created for each Pod.

1. **Select and click on one of Logical ports** that have a CIF in their attachment name
When looking at the port details, you can see how the Address Bindings and Tags section provides information about the Pod that's attached to it.

1. If the Address Bindings or Tags section is closed, click on the arrow to open it.
Review Address Bindings

1. Review the info provided in the Address Bindings Tags section, including MAC Address and IP
1. **Click on the Monitor tab** to see information about the Port itself.

**End of Module**

You have made it to the end of the module. If you want to continue, please go to the next Module.
Module 2 Conclusion

In this module you learned how to create a namespace in K8s and how this integrated with NSX-T. You also created pods and saw how they connected.

You've finished Module 2

Congratulations on completing Module 2.

Proceed to any module below which interests you most.

- **Module 3 - Service and Ingress rules** (30 minutes) (Basic/Advanced) Create and deploy a new POD and update ingress rules
- **Module 4 - Security with NSX-T and Kubernetes** (15 minutes) Leveraging NSX-T to create microsegmentation within Kubernetes

How to End Lab

To end your lab click on the **END** button.
Module 3- Kubernetes Services and Ingress rules (30 minutes)
K8s Services and east-west load balancing using OVS

Kubernetes needs to be able to load balance PODs to handle scale and growth. This section of the lab will provide information on this.

Kubernetes NSX Services

As part of the integration with NSX and K8s, NSX offers improved networking services. One of these services is Kube Proxy.

Upstream Kube Proxy challenge:

- As every POD's network interface is ‘pinned’ down to the Hypervisor vSwitch, it is on purpose bypassing the IP Stack of the Node VM, and with that bypassing IPTables that is used by the K8s Kube Proxy
NSX Kube Proxy:

- Kubernetes Services Cluster IPs are implemented using OpenVSwitch’ new Conntrack (ct) NAT feature. The NSX Kube Proxy daemon is watching the K8s API for new services and is programming flow entries and server groups into OVS.

OVS Load-Balancing:

- East/West Load-Balancing is implemented in the OVS fast path in a fully distributed way.

Module Switcher link

If you are starting this lab from Module 3, you will need to run the Module Switch app to perform some actions that have taken place in Module 2.

1. Click on the Module Switcher link on the desktop
2. Once you see the Module Switcher interface, click on the Module you are starting that lab from.
Module Switcher

Once the script runs, you will need to hit Enter when the script is done.

Now, back to the lab.

Login to K8 Master

1. Click on the Putty Icon on the tool bar
Putty K8Master login

1. Select K8Master from the list
2. Click on Load
3. Click on Open

Default namespace

If you are starting this lab from this Module, please type the following commands to place your user in the proper namespace for the lab. Otherwise, you can ignore the below.

```
kubectl config set-context kubernetes-admin@kubernetes --namespace nsx-ujo
kubectl config use-context kubernetes-admin@kubernetes
```
Review nsx demo service

Now we will take a look at the yaml file that we have created for the nsx service. Type

```
cd demo
vimcat nsx-demo-service.yaml
```

Create the service

To create the service type the following command:

```
kubectl create -f nsx-demo-service.yaml
```

This will return a "service "nsx-demo" created"
Services status

Now to see the status of this newly created service. Type the following to get the information. Your IPs may be different then those in the image.

```
kubectl get svc -o wide
```

1. You can now see that service is running and its Cluster IP.

More details about the service

To see more details about the service and its namespace, ip, etc, type the following:

```
kubectl describe svc/nsx-demo
```

You can see relevant information about the service.

Connect to a node

We need to become root, so type the following commands
Review the OVS flows created

Now we can see the ovs flows created by nsx kube-proxy on OVS for this service. To do this type:

```
svod-ofctl dump-flows br-int
```

1. You can see the different ports that are being seen and flows created for.

**OVS groups**

```
root@k8s-master:/home/localadmin/demo# ovs-ofctl -O OpenFlow13 dump-groups br-int
```

1. Type the following command to see the ovs groups created by nsx kube-proxy on OVS for the service. You will see multiple group numbers.

```
svod-ofctl -O OpenFlow13 dump-groups br-int
```

These grouping are the Pods that are being services by the Load Balancer. Think of it as a load balancing pool. Group ID 1 has the 4 Pods that are being load balanced.
Creating a Pod

1. First, we need to change back to the localadmin user. Do that by typing the following:

   ```bash
   su localadmin
   ```

2. Enter `VMware1!` for the password if prompted

3. Now, we will create a management pod by typing the below command.

   ```bash
   kubectl create -f pod-management.yaml
   ```

   This will return a "pod "mgmtpod" created"

Mgmt Pod status

1. To check the status, type

   ```bash
   kubectl get pods -o wide
   ```

   You can see that the mgmtpod is running and has an IP
**Connect to pod**

We now will connect to the Pod and connect to a website load balanced by the nsx kube-proxy.

1. To do this first type

```
kubectl exec -it mgmtpod /bin/bash
```

This will give you bash access on the POD

**Wget**

Now, let's check if the load balancer is working.

1. To check that the load balancer is working, type the following commands

```
wget -O http://nsx-demo
```

Now you can see the webpage returns. Notice what the containers private IP is.
Run the `wget` command again and see if the containers private IP changes. If it does, then the load balancer is working.

If you receive an error that the **name can not be resolved**, perform the step on the next page, otherwise ignore.

**If you get an error resolving the name**

Type `exit` to leave the bin/bash shell

1. Type the following

   ```bash
   kubectl -n kube-system get pods
   ```

   From the list find the DNS service which status is not currently running.

2. Type the following

   ```bash
   kubectl delete -n kube-system po/kube-dns-xxxxxx
   ```

**NOTE:** You can highlight the number and then left click. This will paste what you have highlighted to the line you are typing.

For the above example it would be:

"kubectl delete -n kube-system po/kube-dns-3913472980-f4kw9"

Wait a few minutes and then try connecting to the `mgmtpod` and running the `wget` command.
Scaling up

What if you want more than 4 Pods. With Kubernetes and NSX is can be done very quickly.

First exit the POD by typing

```
exit
```

1. Now to change the scale number to 6, type the following:

```
kubectl scale --replicas=6 rc/nsx-demo-rc
```

2. Once that is successful type the following to get details about the service and see if it has grown.

```
kubectl describe svc/nsx-demo
```

Changes to OVS group

Now that we have added more endpoints, let check to see the group memberships have been updated in OVS.

1. Type the following to check.
When prompted for a password, type `VMware1!`

1. You will now see that there are 6 members of group_id=1

**Ingress**

Now we need to configure traffic that is coming in externally to our Pod.
K8s Ingress

For this lab, we will test and validate a NGINX Ingress Controller from the upstream K8s Project.

Kubernetes and NSX Ingress

Take a look at the basic lab layout that we have.
Review the Ingress Controller

```
localadmin@k8s-master:~/demo$ vimcat ingress-controller-nginx.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx-ingress-rc
  labels:
    app: nginx-ingress
  annotations:
    ncp-nsx/ingress-controller: "True"
spec:
  replicas: 1
  selector:
    app: nginx-ingress
template:
  metadata:
    labels:
      app: nginx-ingress
  spec:
    # hostNetwork: True
    containers:
    - image: nginxdemos/nginx-ingress:0.3
      imageFullPolicy: Always
      name: nginx-ingress
    #nodeSelector:
    # external: "True"
localadmin@k8s-master:~/demo$
```

From the K8Master, let's take a look at the already deployed controller. Type the following to open the yaml file.

```
vimcat ingress-controller-nginx.yaml
```

**Ready Status?**

```
llocaladmin@k8s-master:~/demo$ kubectl -n default get rc nginx-ingress-rc
NAME         DESIRED  CURRENT  READY AGE
nginx-ingress-rc  1       1        1   2m
localadmin@k8s-master:~/demo$
```

Now to see if the controller is up and running. Type the following:

```
kubectl -n default get rc nginx-ingress-rc
```

As you can see, the controller is Ready.
More details

```yaml
{
  "apiVersion": "v1",
  "kind": "ReplicationController",
  "metadata": {
    "annotations": {
      "kubectl-owner": "controller"
    },
    "name": "nginx-ingress-rc",
    "namespace": "nsx-ujo"
  },
  "spec": {
    "replicas": 1,
    "selector": {
      "app": "nginx-ingress"
    },
    "template": {
      "metadata": {
        "creationTimestamp": null
      },
      "spec": {
        "containers": [
          {
            "image": "nginx:latest",
            "name": "nginx-ingress",
            "imagePullPolicy": "Always",
            "ports": [
              {
                "containerPort": 80,
                "protocol": "TCP"
              }
            ],
            "imagePullPolicy": "Always",
            "terminationMessagePath": "/dev/termination-log",
            "terminationMessagePolicy": "File",
            "restartPolicy": "Always",
            "schedulerName": "default-scheduler",
            "securityContext": {},
            "terminationGracePeriodSeconds": 30
          }
        ]
      }
    }
  }
}
```

Now, let's look at more details about the controller. Type the following

```bash
kubectl -n default get rc nginx-ingress-rc -o yaml
```
Controller network settings

Lastly, we need to make sure the controller is using the nsx specific network settings. Type the following.

```
cat ingress-controller-nginx.yaml | grep ncp-nsx
```

This will result in a "True" response.

NSX Manager login

We now need to check if the dynamically created NAT on the Tier-0 router by NCP. To check this, we need to log back into the NSX Manager. If you have NSX up and logged in, you can skip the following step. Otherwise, perform the following actions.

1. Click on the Google Chrome icon on the tool bar
2. Click on the nsxmgr-01a link in the toolbar

On the NSX Manager login screen fill in the following info.

1. For the User Name, enter: admin
2. For Password, enter: VMware1!
3. Click on Login
NSX Routing

Let's go see what NAT rules are set on the Tier-0 router.

1. Click on the Routing link on the left hand side
We now are going to look at the dynamically created NAT rules on the Tier0 router.

1. Click on NAT tab
2. Click on LogicalRouter-Tier0
3. Look at the NAT rules that are created.

The Nat translated IP is 10.0.2.4. Let's test if it's working

**Testing the NAT**

1. Click on the command prompt icon in the tool bar

```
Pinging 10.0.2.4 with 32 bytes of data:
Reply from 10.0.2.4: bytes=32 time=3ms TTL=63
Reply from 10.0.2.4: bytes=32 time=3ms TTL=63
Reply from 10.0.2.4: bytes=32 time=3ms TTL=63
Reply from 10.0.2.4: bytes=32 time=1ms TTL=63

Ping statistics for 10.0.2.4:
    Packets: Sent = 4, Received = 4, Loss = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 3ms, Average = 2ms
```
2. Type the following ping command to test that the IP of the NAT is working

```
ping 10.0.2.4
```

You will get a successful ping.

**DNS test ping**

![Ping result](image)

The DNS record for the page we are testing is set to allow any word before `demo.corp.local`. Let's see if NAT and DNS are working. Type the following to test.

```
ping blah.demo.corp.local
```

This will reply back with the NAT address of 10.0.2.4

**Review ingress rule**

```
localadmin@k8s-master:$ cd ~/demo
vim nsx-demo-ingress.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nsx-demo-ingress
spec:
  rules:
  - host: nsx.demo.corp.local
    http:
      paths:
      - path: /
        backend:
          serviceName: nsx-demo
          servicePort: 80
```

Return back to your Putty session on the K8 Master. You should still be in the demo folder, but if not type the following

```
cd ~/demo
```
Once back in the demo folder type the following to review the prepared ingress yaml file by typing the following:

```bash
vimcat nsx-demo-ingress.yaml
```

This output shows that the name that host will respond to will be nsx.demo.corp.local on port 80.

### Deploy ingress rule

```
kubectl create -f nsx-demo-ingress.yaml
```

You will receive a "ingress "nsx-demo-ingress" created"

### Review the Ingress rule

```
kubectl describe ingress nsx-demo-ingress
```

Let's see if the ingress rule took. Type the following command:

```bash
ekubectl describe ingress nsx-demo-ingress
```

The output will show the nsx.demo.corp.local is setup to map to the backend servers.
Website test

Hello, this web-page is served by: nsx-demo-rc-d073w

The containers private IP is: 10.4.0.162

As our last test, let's see if the website is working and load balancing.

1. Open a new Tab in Google Chrome
2. Enter nsx.demo.corp.local in the address bar or click on the NSX K8 tab
3. Refresh the page and see the container private IP change.

You should see this page and the successful ingress rules and load balancing working for your lab.

End of Module

This is the end of Module 3. You can now continue on to Module 4 if you want.
Module 3 Conclusion

In this module, you leveraged an external load balancer and then updated the rules to map the external access to your pods.

You've finished Module 3

Congratulations on completing Module 3.

Proceed to any module below which interests you most.

- **Module 4 - Security with NSX-T and Kubernetes** (15 minutes) Leveraging NSX-T to create microsegmentation within Kubernetes

How to End Lab

To end your lab click on the END button.
Module 4 - Security with NSX-T and Kubernetes (15 minutes)
Microsegmentation in K8s with NSX (15 minutes)

Currently in Kubernetes, microsegmentation is very limited. The data model to describe segmentation policies between Namespaces, and within namespaces is called 'Network Policies' and is in beta stage right now. Flannel and many other popular choices don’t support ‘Network Policies’ right now, but the community is working on enhancing the various plugins to support it.

Module Switcher

If you are starting this lab from Module 4 and have not done any other modules, you will need to run the Module Switch app to perform some actions that have been performed in Module 3.

1. Click on the Module Switcher link on the desktop
2. Once you see the Module Switcher interface, click on the Module 3 Start
Module Switcher

Once the script runs, you will need to hit Enter when the script is done.
You need to hit stop **Module 3**
Click on **Module 4 start**, hit enter when the script is done.

Now, back to the lab.

Login to K8Master

If you are already logged into the K8Master, you can skip the next few steps, otherwise please perform the following actions.
1. Click on Putty icon on the task bar

**Putty to K8**

1. Select K8Master from the list
2. Click on Load
3. Click on Open

**Default Namespace**

If you are starting this lab from this Module, please type the following commands to place your user in the proper namespace for the lab. Otherwise, you can ignore the below.

```
kubectl config set-context kubernetes-admin@kubernetes --namespace nsx-uyo
kubectl config use-context kubernetes-admin@kubernetes
```

**Delete existing nsx-demo if existing**

```
localadmin@k8s-master:/demo$ kubectl delete rc/nsx-demo-rc
replicationcontroller "nsx-demo-rc" deleted
```

We need to delete any existing nsx-demo replication controllers. To perform this, enter the following commands.
kubectl delete rc/nsx-demo-rc

This should return a "replicationcontroller 'nsx-demo-rc" deleted." If you receive a "not found" respond, please ignore and continue on.

Review security controller

```yaml
localadmin@k8s-master:~/demo$ vim cat nsx-demo-controller-secgroup.yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nsx-demo-rc
  labels:
    app: nsx-demo
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nsx-demo
    spec:
      containers:
        - name: nsx-demo
          image: yfauser/nsx-demo
          ports:
            - containerPort: 80

```

Now we are going to create a new security replication controller. This is located in the Demo folder. Lets get to the demo folder and review the file. Type the following

```
cd ~/demo
vim cat nsx-demo-controller-secgroup.yaml
```

As you can see in the yaml file we are using a term of secgroup: web-tier. This is different than the standard controller. To see the difference run the following command on the standard nsx-demo controller by typing

```
vim cat nsx-demo-controller-no-labels.yaml
```

Deploy sec controller

```
llocaladmin@k8s-master:~/demo$ kubectl create -f nsx-demo-controller-secgroup.yaml
replicationcontroller "nsx-demo-rc" created
```
Now, deploy the new controllers with the secgroup tag by typing the following command:

```
kubectl create -f nsx-demo-controller-secgroup.yaml
```

You will see a replicationcontroller "nsx-demo-rc" created.

**Launch Chrome**

1. Minimize the putty session and click on Google Chrome from the tool bar if its not already open.
2. Click on the nsxmgr-01a link in the toolbar.
Login into NSX

1. For the User Name, enter admin
2. For Password, enter: VMware1!
3. Click on Login
Switch port

1. Click on the Switching link on the left hand side.

Logical Switch and port
1. Select the k8s-cl1-nsx-ujo switch
2. Click on the Related tab > Ports
3. Select and click on a Logical Port that has VIF in its Attachment name. Any of the ports will work, as long as the name starts with CIF.

Ports and tags

1. Now, open the Tags section
2. Notice that the secgroup is carried over from the deployment

NSX can use this attribute to form groups which can be used in the DFW.
Group

Click on the Inventory link on the left hand side to go to Groups
Web-tier Group

1. Click on the web-tier group
2. Click on the Membership Criteria tab

Here you see how NSX is defining the members of the group. We are looking for a Tag that is equal to web-tier

Web-Tier Members

1. Click on the Members tab
2. Select Logical Port as the Object Type

Now, let's look at the NSX Group and who is a member

1. Click on the Members tab
2. Select Logical Port as the Object Type
You will now see all the members of the Group. These are the ports used by the deployed nsx controller we deployed earlier.

1. Click on Effective
2. Select one of the ports and click on it
Port review

1. Now, open the Tags section if it's not open
2. You can see that this port is part of the secgroup web-tier.
NSX-T has a new search feature which comes in handy when looking for objects. Let's find all the ports that are part of the secgroup.

1. Click on the search icon in the upper right side
2. Type secgroup

See how the logical ports automatically start to appear.
Now, let's look at the firewall rules that are leveraging this Group to provide microsegmentation.

Click on the Firewall link on the left hand side.
DFW rule

We see an existing rule that leverages the web-tier group we were just looking at. The rule states that any member of the web-tier group cannot communicate with any other members of the group. Complete isolation from each other.

Web page test

Hello, this web-page is served by: nsx-demo-rc-z925g
The containers private IP is: 10.4.0.162
Now let's see if this work.

1. Open a new tab in Google Chrome
2. Type the following url

```
nsx.demo.corp.local
```

3. Copy container private IP. You will need to use it in the next step.
4. Click on the Secret port scan app

This will open a new page that we can use to test the firewall rule.

**Port Scan**

![Port Scan App](image)

1. Change the network to scan range to 10.4.0.160/29
2. Click Send

**Scan results**

1. As you see the results come in, you see that the only machine that will respond with an open port is the Pod you ran the scan from. (Your scan may give different results depending on which server launched the scan.)
The DFW is blocking all traffic between the PODs even though they share the same network and allowing to do microsegmenation.

2. Why do we see a successful scan of an IP but on port 22 and not 80?

**Pod IPs**

```
localadmin@k8s-master:/demo$ kubectl get pods -o wide
NAME     READY STATUS    RESTARTS AGE   IP                NODE
mgmtpod  1/1   Running 0 51m  10.4.0.166  k8s-node1
nsx-demo-rc-dhzt1  1/1   Running 0 10m  10.4.0.165  k8s-node1
nsx-demo-rc-dlg0x  1/1   Running 0 10m  10.4.0.163  k8s-node1
nsx-demo-rc-mcrjt  1/1   Running 0 10m  10.4.0.162  k8s-node2
nsx-demo-rc-xr15h  1/1   Running 0 10m  10.4.0.164  k8s-node2
localadmin@k8s-master:/demo$
```

If you run the following command, you will see that the mgmtpod has the IP that was seen on port 22. Since it's not part of the security group, it is accessable.

```
kubectl get pods -o wide
```

1. Here you see the IP of the mgmtpod.

**End**

Thanks for taking this lab. We hope it was informative and enjoyable. If you want to learn more about NSX-T, please take a look at lab 1826-01.
Module 4 Conclusion

In this module, you saw how NSX security can be leveraged to create microsegmentation.

You've finished Module 4 and this lab.

Congratulations on completing Module 4 and the lab. We hope you enjoyed it. If you are interested in doing another Kubernetes related lab, please go see HOL-1831-01-CNA.

How to End Lab

To end your lab click on the END button.
Conclusion

Thank you for participating in the VMware Hands-on Labs. Be sure to visit http://hol.vmware.com/ to continue your lab experience online.

Lab SKU: HOL-1826-02-NET

Version: 20180425-080935