NX-OSv 9000

This chapter contains the following sections:

About NX-OSv 9000

The NX-OSv 9000 is a virtual platform that is designed to simulate the control plane aspects of a network element running Cisco Nexus 9000 software. The NX-OSv 9000 shares the same software image running on Cisco Nexus 9000 hardware platform although no specific hardware emulation is implemented. When the software runs as a virtual machine, line card (LC) ASIC provisioning or any interaction from the control plane to hardware ASIC is handled by the NX-OSv 9000 software data plane.

The NX-OSv 9000 for the Cisco Nexus 9000 Series provides a useful tool to enable the devops model and rapidly test changes to the infrastructure or to infrastructure automation tools. This enables customers to validate configuration changes on a simulated network prior to applying them on a production network. Some users have also expressed interest in using the simulation system for feature test ,verification, and automation

tooling development and test simulation prior to deployment. NX-OSv 9000 can be used as a programmability vehicle to validate software defined networks (SDNs) and Network Function Virtualization (NFV) based solutions.

NX-OSv 9000 Guidelines and Limitations

NX-OSv 9000 has the following guidelines and limitations:

- After initial setup of the NX-OSv 9000, you must configure the booting image in your system. Otherwise, the NX-OSv 9000 drops to the `loader>` prompt after reload/shut down.

```
switch# configure terminal
switch(config)#  boot nxos n9000-dk9.7.0.3.I2.0.454.bin
switch(config)#  copy running-config startup-config
```

- The NX-OSv 9000 uses vNICs that are entered from the KVM/QEMU command line or from the GUI on ESXi for networking either externally or internally within a hypervisor server. The first NIC is always used as the NX-OSv 9000 management interface. The subsequence NICs are used as data ports as e1/1, e1/2, ... e1/9. Ten NICs are allowed with nine NICs for data ports.

  Note Beginning with Cisco NX-OS Release 7.0(3)I5(2), a maximum of 64 data ports (e1/1, e1/2, ... e1/64) are supported.

  Connect only the first NIC for the NX-OSv 9000 VM as the management interface to your LAN physical switch or vSwitch (VM Network) connecting directly to a physical switch. Do not connect any data port vNIC to any physical switch that conflicts with your server management connectivity.

- NX-OSv 9000 only supports the ESXi standard vSwitch when VMs are interconnected within a hypervisor or an external physical switch.
- The vSwitch mapping to data port interface is required to have Promiscuous Mode as the Accept mode in order to pass traffic between VMs.
- Multicast snooping should be disabled on net-devices configured as bridges on Linux-host machines. These net-devices are used to connect VMs in the NX-OSv 9000 setup.
- The NX-OSv 9000 operates as a bridge that generates BPDU packets on its Ethernet interfaces as it participates in Spanning Tree Protocol (STP). It also forwards broadcast, unknown unicast, and multicast traffic as expected by classic bridging logic. Do not connect the NX-OSv 9000 data plane interfaces to the upstream network in a manner that would create bridging loops or interfere with upstream STP operation.
- Beginning with Cisco NX-OS Release 7.0(3)I6(1), NX-OSv 9000 is supported in the Virtual Internet Routing Lab (VIRL) and the Cisco Modeling Lab (CML) environment running as a VM.
- Beginning with Cisco NX-OS Release 7.0(3)I6(1), VXLAN BGP EVPN is supported on NX-OSv 9000.

Benefits of Virtualization Using the NX-OSv 9000

This virtual platform provides these virtualization benefits in a cloud environment and you are not limited to the type of hardware as well as other resources.

Benefits	Description
Hardware Independence	This virtual platform provides these virtualization benefits in a cloud environment and users is not limited to hardware as well as other resources. **Note** The minimum RAM/memory requirement for an NX-OSv 9000 based VM is 4GB
Resource Sharing	The resources used by NX-OSv 9000 are managed by the hypervisor, and can be shared among VMs. The amount of hardware resources that VM sever allocates to a specific VM, can be reallocated to another VM on the server.
Flexibility in Deployment	You can easily move a VM from one server to another, Thus, you can move the NX-OSv 9000 from a server in one physical location to a server in another physical location without moving any hardware resources.
Dynamic Networking	Users can change network connectivity and configuration in a matter of mins without any physical cabling.

NX-OSv 9000 Software Functionality

NX-OS 7.0(3)I6(1) Release and Earlier Releases

Beginning Cisco NX-OS 7.0(3)I5(1), NX-OSv 9000 supports emulation and implementation on a subset of hardware dependent features. Enablement of such features in the forwarding plane allows users to simulate a more realistic deployment scenario such as an NX-OS hardware platform.

The following are restrictions and host environment differences for NX-OSv 9000:

- Interface throughputs are currently rate-limited to prevent any production usage. The interface rate-limit is set as 4M per device.
- For the Oracle VM VirtualBox hypervisor, the SATA block device must be used due to the image size increase from the 7.0(3)I5(1) branch which impacts the legacy bios IDE controller size limit. SATA block devices also significantly improve disk performance, especially for initial boot up time. The IDE controller can be used in the VMware ESXi and the KVM/QEMU hypervisor, but it is significantly slower than the SATA controller.
- A SATA block device can also be used in KVM/QEMU hyerpvisor, but it requires QEMU 2.2.0 or later version. Similar to the Oracle VM VirtualBox hypervisor, a performance improvement can be obtained in the KVM/QEMU environment.

Supported Features

The following table displays specific Layer 2 and Layer 3 software feature support based on branch/lineup.

Table 1: Supported Layer 2 and Layer 3 Features (Software)

Technology Branch/Lineup	Nexus Feature Name	Support Statement Cisco NX-OS 7.0(3)I5(1), 7.0(3)I5(2), and 7.0(3)I6(1) Releases
OS Infra	Bash Shell	Supported
	Guest Shell	Supported
	SSH	Supported
	RPM Installation	Supported
	POAP	Supported for the management interface in Cisco NX-OS Release 7.0(3)I5(1) and for all interfaces in Cisco NX-OS Release 7.0(3)I5(2) and Cisco NX-OS Release 7.0(3)I6(1)
Programmability	NXAPI	Supported
	Puppet Integration (Native)	Supported
	Puppet Integration (Guest Shell)	Supported
	Chef Integration (Native)	Supported
	Chef Integration (Guest Shell)	Supported
L3 Features	CDP	Supported
	LLDP	Supported
	BGP v4	Supported (No BFD, EVPN) **Note** VXLAN EVPN is supported in Release7.0(3)I6(1) and later.
	BGP v6	Supported (No BFD, EVPN)
	OSPFv2	Supported (No BFD, EVPN)
	OSPFv3	Supported (No BFD, EVPN)
	EIGRP	Supported
	RIP	Supported

Technology Branch/Lineup	Nexus Feature Name	Support Statement Cisco NX-OS 7.0(3)I5(1), 7.0(3)I5(2), and 7.0(3)I6(1) Releases
L2 Features	L2 Switching Unicast	Supported
	L2 Switching Broadcast	Supported
	L2 Switching Multicast	Supported as Broadcast (not explicit Mcast) , No PIM or Mcast Group support
	MAC learning	Supported
	Static/Router MAC	Supported
	Switchport	Supported
	1. 802.1q Trunk	Supported
	2. 802.1q Access	Supported
	STP	Supported
	L3 SVI	Supported
	Subinterfaces	Supported
	VXLAN	Supported (Flood and Learn); BGP EVPN without ARP suppression in Cisco NX-OS Release 7.0(3)I6(1) and later.
	vPC	Not supported in Cisco NX-OS Release 7.0(3)I5(1), but supported in Cisco NX-OS Release 7.0(3)I5(2) and Cisco NX-OS Release 7.0(3)I6(1)
	Port channel	Supported

Note The NX-OSv 9000 features in this table have been verified to operate only with the Cisco devices mentioned in this document.

If a networking or system feature is not identified as a supported feature in this document, it should be considered as unsupported despite that it may seem to work correctly. Unsupported features did not have any level of regression testing on NX-OSv 9000.

Table 2: NX-OS Features Not Supported (Not Tested)

NX-OS Features	Limitations
QoS	Not supported on NX-OSv 9000.
BFD	Not supported on NX-OSv 9000.
ACL	Not supported on NX-OSv 9000.
Policy maps	Not supported on NX-OSv 9000.
ARP Suppression	Not supported on NX-OSv 9000.
SPAN	Not supported on NX-OSv 9000.
IGMP Snooping	Not supported on NX-OSv 9000.
AMT	Not supported on NX-OSv 9000.
LISP	Not supported on NX-OSv 9000.
OTV	Not supported on NX-OSv 9000.

The following list (not comprehensive) contains known system limitations.

Table 3: NX-OS System Limitations

System Capabilities	Limitations
MAC Address	NX-OSv 9000 does not integrate the L2FM module and L2FDWR data plane. It maintains its own MAC Table. Therefore the behavior of the MAC address related CLIs will be different from the physical platform.
Statistics	NX-OSv 9000 re-uses a software data plane that was created for L2FDWR. With this data plane, NX-OSv 9000 will not be able to account for and display traffic related statistics (such as interface statistics).
Consistency Checker	The consistency checker has a hardware dependency and hence is not supported on NX-OSv 9000. All ‘show’ and ‘exec’ commands will result with appropriate error/warnings.
Network Throughput	Low data plane performance. Additional rate limiter is in place to limit the total amount of traffic received by NX-OSv 9000 to 4M.
TOR-ISSU	TOR-ISSU is not supported.
Link Status	NX-OSv 9000 virtual interfaces serve as the ‘Ethernet Ports’. The link status of these links within the NX-OS is dependent on the Hypervisor’s capability.

System Capabilities	Limitations
Link-down	Connectivity between the two ends of the interface link is simulated, hence it is important that you shut the interface in both the ends, followed by no shut at both the ends of the interface link.

NX-OSv 9000 Feature UI/CLI Difference From Hardware Platform

Feature enablement in the NX-OSv 9000 virtual platform is the same as Cisco Nexus 9000 hardware platform.

For example, the following features can be enabled:

- **feature telnet**
- **feature bash-shell**
- **feature ospf**
- **feature bgp**
- **feature interface-vlan**
- **feature nv overlay**

However, not all commands are available for NX-OSv 9000, such as hardware data plane specific commands. Some of these commands exist in the command parse chain, but these commands might not display correct output information. It is not possible for the virtual platform to verify all commands on NX-OSv 9000 that exist for the Cisco Nexus 9000 hardware platform.

A few commands are critical for NX-OSv 9000 to display Layer 2/Layer 3 information, but are not provided for the NX-OSv 9000 platform. The following displays substitute commands:

NX-OS Hardware Platform Commands	Substitute for NX-OSv 9000
show mac address-table	**show system internal l2fwder mac**
clear mac address-table	**clear mac address-table datapath static|dynamic**

NX-OSv 9000 Resource Requirements

The NX-OSv 9000 uses the Cisco Nexus 9000 Series hardware software image. It requires the minimum resources as shown in the following list. These resources are generally not oversubscribed on any server.

- 8G memory
- Minimum 4G. We recommend a 8G VM configuration for complex topology and enabling features.
- 1-4 vCPUs
- 4G hard disk
- 1 serial port
- 1 network interface card (NIC)

Server Software Requirements

The NX-OSv 9000 can run on Cisco Unified Computing System (UCS) servers or servers from leading vendors that support VMware ESXi 5.1 (Post Build 1065491/ ESXi 5.5) or the combination of Ubuntu Linux 14.04LTS or later version and KVM-QEMU 2.0.

if you only need a standalone NX-OSv 9000 node, the NX-OSv 9000 can also be deployed on a laptop or and Apple Mac Pro with a virtual box hypervisor as long as your laptop meets basic resource requirements.

VMware ESXi Support Information

The virtual machine (VM) runs on the VMware vSphere Hypervisor. You can use the same VMware vSphere hypervisor to run serial VMs. Use the VMware vSphere Client GUI to create and manager VMs.

The VMware vSphere Client is an application for creating, configuring, and managing VMs on the VMware vCenter Server. The NX-OSv 9000 can boot from a virtual disk located on the data store. You can perform basic administration tasks such as starting and stopping the NX-OSv 9000, using the VMware vSphere Client.

VMWare vCenter Server manages the vSphere environment and provides unified management of all the hosts and VMs in the data center from a single console.

For more information about how Cisco and VMware work together, see http://www.vmware.com/cisco.

For more information about VMware features and operations, see the https://www.vmware.com/support/pubs/ .

KVM-QEMU Support Information

The kernel-based Virtual Machine (KVM) is an open-source, full-virtualization solution for Linux on x86 hardware, containing virtualization extensions. It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor-specific module, ivm-intel.ko or kvm-amd.ko

Quick Emulator (QEMU) is a free and open-source software product that performs hardware virtualization. You can run QEMU on the Cisco UCS server with KVM installed. The recommended version of QEMU for the NX-OSv 9000 reference platform is version 2.2.0 or later.

VirtualBox Support Information

VirtualBox is a powerful x86 and AMD64/Intel 64 virtualization product for enterprise as well as for the home user. It is free software available as Open Source Software under the terms of the GNU General Public License (GPL) version 2 and you can obtain more information and download from https://www.virtualbox.org/ web site.

VMware Fusion Support Information

VMware Fusion is also a powerful virtualization product for enterprise as well as PC user.

NX-OSv 9000 Installation/Deployment

NX-OSv 9000 currently does not support virtio block disk. To optimize performance, specific virtual artifact formats are recommended to be used in particular hypervisor.

Hypervisor	Virtual Artifact Format
EXSi	Virtual Machine Disk Image (vmdk), Open Virtualization Appliance (ova)
KVM/Qemu	QEMU Copy On Write (qcow2),Open Virtualization Appliance (ova)
Virtual Box	Virtual Machine Disk Image (vmdk), or packaged box
VMware Fusion	Open Virtualization Appliance (ova)

NX-OSv 9000 Software Upgrade and Downgrade

The software upgrade and downgrade of NX-OSv 9000 does not follow normal hardware platform procedures. A common upgrade method for NX-OSv 9000 is to tftp or scp a new image into the bootflash, then boot the new image from the loader> prompt or set the boot image in "config t; boot nxos bootflash:new_image.bin". A similar approach is used for downgrade.

Note This approach requires sufficient bootflash disk space to hold another image. As such, the nxos.7.0.3.I2.2a image is not upgradable to a new release. In this case, you can create a new VM based on the nxosv-final.7.0.3.I2.2d release; and then upgrade to a new release.

NX-OSv 9000 Configuration

Beginning with Cisco NX-OS Release 7.0(3)I5(2), Cisco NX-OSv 9000 supports the Cisco Virtual Appliance Configuration (CVAC). This out-of-band configuration mechanism is similar to the POAP autoconfiguration, but instead of downloading the configuration across the network as POAP does, CVAC receives the configuration injected into the Cisco NX-OSv 9000 environment on a CD-ROM. The configuration is detected and applied at startup time.

CVAC can be used for a bootstrap configuration (supplying just enough configuration to bring the switch into a reachable state suitable for subsequent configuration using Telnet, RESTful APIs, or other standard mechanisms) or a full configuration (taking the entire configuration of another router and replicating it into a newly launched platform VM). The configuration should be in a plain-text file called nxos_config.txt. You can package the configuration file onto the CD-ROM using the following command:

```
mkisofs -output nxosconfig.iso -l --relaxed-filenames --iso-level 2 <file(s) to add>
```

If the system does not detect a CVAC configuration, the POAP process begins, and the POAP interface prompts you for the initial installation. See PowerOn Auto Provisioning for a newly installed switch.

The Cisco NX-OSv 9000 supports the same control plane features and configuration that are supported on the Cisco Nexus 9000 Series hardware platforms. The configuration commands for the control plane features follow the same syntax as the Cisco Nexus 9000 Series switches.

NX-OSv 9000 Deployment

Provisioning the NX-OSv 9000 in the ESXi Hypervisor

Before You Begin

You must have already installed your ESXi hypervisor.

Step 1 Log into the VMware vSphere Client.

Step 2 Select your server's IP address and click the **Create a new virtual machine** link.

Step 3 In the **Configuration** screen, click the **Custom** button and click **Next**.

Step 4 In the **Name and Location** screen, enter your virtual machine's name and click **Next**.

Step 5 In the **Storage** screen, select the datastore and click **Next**.

Step 6 In the **Virtual Machine Version** screen, choose the **Virtual Machine Version** button and click **Next**.

Step 7 In the **Guest Operating System** screen, choose Other (64-bit), Version, and click **Next**.

Step 8 In the **CPUs** screen, choose Number of cores per virtual socket value and click **Next**.

Step 9 In the **Memory** screen, enter the memory size and click **Next**.

Note The minimum size is 4 GB, but the preferable size is 8 GB.

Step 10 In the **Network** screen, select the number of NICs you want to connect.

Note The first NIC must be the management interface. Do not connect all dataports to the management interface.

Step 11 In the **SCSI Controller** screen, choose your SCSI controller, and click **Next**.

Step 12 In the **Select a disk** screen, choose the **Use an existing virtual disk** button, and click **Next**.

Step 13 In the **Select Existing Disk** screen:

a) Browse to the location of the n9000v-dk9-final.7.0.3.I2.1.vmdk file and click **OK**.

b) Click **Next**.

Step 14 In the **Advanced Options** screen, confirm that the Virtual Device Node is IDE (0:0), and click **Next**.

Step 15 In the **Ready to Complete** screen, chose the **Edit the virtual machine settings before completion** check box, and click **Continue**.

Step 16 In the **Virtual Machine Properites** screen, click the **Add** button to add a Serial Port, confirm that the Virtual Device Node is IDE (0:0), and click **Next**.

Step 17 In the **Device Type** screen, choose Serial Port and click **Next**.

Step 18 In the **Serial Port Type** screen, choose the **Connect via Network** button, and click **Next**.

Step 19 In the **Network Serial Port Settings** screen:

a) In the **Network Backing** frame, choose the **Server** button.
b) Enter telnet and the server IP address in the **Port URI** field, and click **Next**.

Step 20 In the **Ready to Complete** screen, click **Finish**.

Step 21 In the **Getting Started** tab, click the **Edit virtual machine settings** link.

Step 22 In the vSphere Client, select the name of the virtual machine and click the **Options** tab.

Step 23 In the **Virtual Machine Properties** screen, choose Boot Options in the Setting column.

Step 24 In the **Firmware** panel, choose the **EFI** button and click **OK.**

Step 25 Power on the virtual machine.

NX-OSv 9000 Deployment on KVM/QEMU Hypervisor

The NX-OSv 9000 can be brought up in the KVM/QEMU hypervisor. The following table lists the parameters that are supported for the NX-OSv 9000 deployment on KVM/QEMU.

Parameter	Example	Description
/path_to/qemu	/usr/bin/qemu-system-x86_64	Path to QEMU executable. (The QEMU software can be downloaded from http://wiki.qemu.org/download for different versions.)
-nographic	-nographic	Recommended, as the NX-OSv 9000 does not support VGA.
-bios file	-bios bios.bin	Required. The NX-OSv 9000 uses EFI boot and requires a compatible BIOS image to operate. We recommend using the latest OVMF BIOS file with the SATA controller for better performance in terms of disk operation. QEMU 2.6 is recommended with the SATA controller. For more information, see http://www.linux-kvm.org/page/OVMF.
-smp	-smp 4	The NX-OSv 9000 supports one to four vCPUs, but two to four are recommended.

Parameter	Example	Description
-m memory	-m 8096	Minimum memory is required.
-serial telnet:host:port,server,nowait	-serial telnet:localhost:8888,server,nowait or -serial telnet:server_ip:8888,server,nowait	Requires at least one.
-net ... -net ... or -netdev ... -device ...	-net socket,vlan=x,name=nl_s0,listen=localhost:12000 -net nic, vlan=x,model=e1000,macaddr=aaaa.bbbb.cccc -netdev socket,listen=localhost:12000,id=eth_s_f -device e1000,addr=s.f,netdev=eth_s_f, mac=aaaa.bbbb.cccc,multifunction=on,romfile= or -netdev tap,ifname=tap_s_f,script=no, downscript=no,id=eth_s_f -device e1000,addr=s.f,netdev=eth_s_f, mac=aaaa.bbbb.ccc,multifunction=on,romfile=	The net/net or netdev/device pairs are for networking a virtual network interface card (vNIC). The _s_f represents the PCI slot number and function number. QEMU 2.0 or above has the capability to plug in at least 20 PCI slots and four functions, which accommodates about 80 vNICs in total. The slot range is from 3 to 19, and the function number range is from 0 to 3. The mac= option passes the MAC address of each vNIC MAC address to the VM interfaces. The first -netdev is automatically mapped to the mgmt0 interface on the VM. The second -netdev is mapped to the e1/1 interface and so on up to the sixty-fifth on e1/64. Make sure these MAC addresses are unique for each network device.
-enable-kvm	-enable-kvm	This flag is required for the NX-OSv 9000.

Parameter	Example	Description
-drive ... -device ... (for the SATA controller)	-device ahci,id=ahci0,bus=pci.0 -drive file=img.qcow2,if=none,id=drive-sata-disk0,format=qcow2 -device ide-drive,bus=ahci0.0,drive=drive-sata-disk0,id=drive-sata-disk0	Format to use the SATA controller. We recommend using the SATA controller with QEMU 2.6.0 because this controller offers better performance than the IDE controller. However, you can use the IDE controller if you have an early QEMU version that does not support the SATA controller.
-drive ... -device .... (for the IDE controller)	-drive file=img.qcow2,if=none,id=drive-ide0-0-0,format=qcow2 -device ide-hd,bus=ide.0,unit=1,drive=drive-ide0-0-0,id=ide0-0-0	Format to use the IDE controller.
-drive ... media=cdrom	-drive file=cfg.iso,media=cdrom	CD-ROM disk containing a switch configuration file that will be applied after the NX-OSv 9000 comes up. 1. Name a text file (nxos_config.txt). 2. Use Linux commands to make cfg.iso, mkisofs -o cfg.iso -l --iso-level 2 nxos_config.txt.

KVM/QEMU Environment Networking

Choosing a Networking Transport Type

QEMU/KVM supports several different VM network transport types:

- Socket transport
- Tap and bridge transport
- User transport

Each type has advantages and disadvantages. Depending on your host access and the devices or networks to which you wish to connect your VM, some transports may be more convenient than others.

Table 4: Transport Type Advantages and Disadvantages

Transport Type	Advantages	Disadvantages
Socket	• Does not require sudo/root access to set up • Decent performance	• Only point-to-point connectivity is possible (no LAN capability) • Must bring up VMs in order ("listen" socket before "connect") • Can lose connectivity if VMs are stopped and restarted • Can connect only to other VMs, not to a host or lab network
Tap and bridge	• Can connect multiple VMs to a single LAN • Can connect VMs to a host or to other devices on your lab network • Can run **tcpdump** (or Wireshark) to capture traffic on tap interfaces • Generally the best performance	• Requires sudo/root access to create taps and bridges • Misconfiguration can render the host unreachable, requiring a reboot of the host
User	• Does not require sudo/root access to set up • Can connect VMs to a host	• Very slow performance • Some protocols (for example, ICMP and ping) are not supported • Cannot connect to a lab network or to other VMs, only to a host

General guidelines:

- If you have root access to create taps and bridges, tap and bridge transport is the most flexible and powerful transport type, but it requires caution during setup.
- If you do not have root access and want to interconnect VMs, socket transport is the fallback option.

- If you do not have root access and want to connect the VM to the host, user transport will work but has limitations.
- If you have no need for network connectivity, you do not need a transport at all (see Networking CLI Commands).

Networking CLI Commands

You have two ways to add NICs using the QEMU CLI. One (-net/-net) is simpler but officially deprecated. The other (-netdev/-device) is the current recommendation but is slightly more complex. In both cases, two QEMU commands are required per NIC. One defines the NIC's appearance to the NX-OSv 9000 (primarily its MAC address), and the other defines the underlying transport mechanism.

Task	Old CLI	New CLI
Create NIC	-net nic,model=*model*,vlan=*number*,macaddr=*macaddr* **Note** Typical values for *model* when using the old CLI are virtio or e1000.	-device *model*,netdev=*string*,macaddr=*macaddr* **Note** Typical values for *model* when using the new CLI are virtio-net-pci or e1000.
Define network transport for NIC	-net *transport*,*options*,vlan=*number*	-netdev *transport*,*options*,id=*string*

If your VM does not require connectivity to other hosts or VMs, you can omit the -net *transport*... or -netdev *transport*... commands entirely. As a result, NICs are visible to the guest and configurable for testing but do not pass packets.

With the old CLI, the vlan=*number* parameter is used to link the two commands together. (Both commands for a NIC should use the same *number*.) With the new CLI, the netdev=*string* and id=*string* parameters serve this purpose. (The *string* values must match.)

Socket Transport

Point-to-point connections between VMs can use TCP sockets as the transport mechanism. One VM will open a listen socket, and the other VM will connect to this socket.

Task	Old CLI	New CLI
First VM	-net socket,listen=*host1*:*port*,...	-netdev socket,listen=*host1*:*port*,...
Second VM	-net socket,connect=*host1*:*port*,...	-netdev socket,connect=*host1*:*port*,...

In socket transport, the listen VM must be started before the connect VM; otherwise, the connection between VMs will not be successfully established. Similarly, if the listen VM is terminated and relaunched, you might need to terminate and relaunch the connect VM, which can be challenging in complex topologies.

Note In QEMU versions prior to 1.4, you cannot use -netdev socket as it will fail with the following error: Property 'model.netdev' can't find value 'netdev'. If you are using QEMU 1.0 or 1.2.x, you must use -net for socket transports.

Tap and Bridge Transport

The host can provide tap interfaces connected to a virtual bridge in order to provide LAN connectivity between multiple VMs. Each VM NIC uses an individual tap interface as its designated transport, and the host connects the taps to a shared bridge as needed.

The *bringup-script* and *teardown-script* parameters are pointers to scripts used to dynamically create and tear down the tap interfaces. If you are using previously created tap interfaces, set these parameters to "no" to indicate that no such script is needed.

You can use the **tcpdump** command to capture traffic on tap interfaces, which can be very useful for troubleshooting.

Connecting Two VMs

You can connect VMs to each other by adding the appropriate tap interfaces for each VM to a logical Ethernet bridge.

Before You Begin

You need sudo access to run the commands in this procedure.

DETAILED STEPS

	Command or Action	Purpose

Step 1	**openvpn --mktun --dev**
	
	Example:
	`sudo openvpn --mktun --dev mytap0`
	`sudo openvpn --mktun --dev mytap1`
	`sudo ifconfig mytap0 up`
	`sudo ifconfig mytap1 up`
		Creates two tap interfaces.

Step 2	**brctl addbr** *bridge-name* **Example:** `$ sudo brctl addbr mybridge0`	Creates a bridge.

Step 3	**brctl addif** *bridge-name tap-name*
	
	Example:
	`$ sudo brctl addif mybridge0 mytap0`
	`$ sudo brctl addif mybridge0 mytap1`
		Adds tap interfaces to the bridge.

Step 4	**ifconfig** *bridgename* **up** **Example:** `$ sudo ifconfig mybridge0 up`	Ensures that the bridge is up and running to bridge the traffic.

	Command or Action	Purpose

Step 5	**brctl show**
	
	Example:
	`bridge name  bridge id           STP enabled   interfaces`
	`mybridge0    8000.1217b6e4a8f3   no            mytap0 mytap1`
	`$`
		Verifies the bridge setup.

Step 6	**qemu-system-x86_64**
	
	Example:
	`$ qemu-system-x86_64 ... -net tap,ifname=mytap0,script=no,downscript=no &`
	`$ qemu-system-x86_64 ... -net tap,ifname=mytap1,script=no,downscript=no &`
	
	or
	
	`$ qemu-system-x86_64 ... -netdev tap,id=mytap0,ifname=mytap0,script=no,downscript=no \  -device e1000,netdev=mytap0,mac=00:10:20:30:40:50 &`
	`$ qemu-system-x86_64 ... -netdev tap,id=mytap1,ifname=mytap1,script=no,downscript=no \  -device e1000,netdev=mytap1,mac=00:11:22:33:44:55 &`
		Starts VMs using tap transport.

Simulating a Large Number of Network Interfaces on KVM/QEMU

You can increase the number of network interfaces for a guest VM by specifying the PCI address in the -device option.

In the following example, users should have 1+2 network interfaces in the NX-OSv 9000 guest. The first vNIC maps to the NX-OSv 9000 management interface. The second and third vNICs map to the NX-OSv 9000 e1/1 and e1/2 data ports, respectively.

Note Make sure that you download the latest BIOS image as well as QEMU 2.2 or a later version when you use the following examples.

VM1

```
qemu-system-x86_64 -smp 4 -enable-kvm -m 8192 -bios -nographic -serial telnet:localhost
12000,server,nowait \

   -netdevice  socket,listen=locahost:13000,id=eth_3_0 -device
e1000,addr=3.0,netdevice=eth_3_0,mac=00:01:80:ff:00:00 \

   -netdevice  socket,listen=locahost:13001,id=eth_3_1 -device
e1000,addr=3.1,netdevice=eth_3_1,mac=00:01:80:ff:00:01 \

   -netdevice  socket,listen=locahost:13002,id=eth_3_2 -device
e1000,addr=3.2,netdevice=eth_3_2,mac=00:01:80:ff:00:02 \

   -device ahci,id=ahci0,bus=pci.0 -drive
file=r1.nxosv-final.7.0.3.I5.1.qcow2,if=none,id=drive-sata-disk0,id=drive-sata-disk0,format=qcow2
 \

   -device ide-drive,bus=ahci0.0,drive=drive-sata-disk0,id=drive-sata-disk0
```

VM2

```
qemu-system-x86_64 -smp 4 -enable-kvm -m 8192 -bios bios.bin -nographic -serial
telnet:localhost 12100, server,nowait -netdevice

   -netdevice  socket,connect=locahost:13000,id=eth_3_0 -device
e1000,addr=3.0,netdevice=eth_3_0,mac=00:01:80:ff:00:10 \

   -netdevice  socket,connect=locahost:13001,id=eth_3_1 -device
e1000,addr=3.1,netdevice=eth_3_1,mac=00:01:80:ff:00:11 \

   -netdevice  socket,connect=locahost:13002,id=eth_3_2 -device
e1000,addr=3.2,netdevice=eth_3_2,mac=00:01:80:ff:00:12 \

   -device ahci,id=ahci0,bus=pci.0 -drive
file=r2.nxosv-final.7.0.3.I5.1.qcow2,if=none,id=drive-sata-disk0,id=drive-sata-disk0,format=qcow2
 \

   -device ide-drive,bus=ahci0.0,drive=drive-sata-disk0,id=drive-sata-disk0
```

NX-OSv 9000 Management Connectivity External to the KVM Hypervisor Server

The general approach to having VM connectivity to outside the device is to pass a tap (in QEMU/KVM command invocation) that is connected to a physical interface on the server through a bridge. The first tap interface -netdev in QEMU/KVM command invocation is mapped to the management interface on VM, mgmt0. We recommend using the secondary physical port in the KVM server instead of the primary physical port, which services the connectivity to this device. Doing so will prevent connectivity issues from the server to any users.

The following example shows how to connect the VM management interface using net device eth1 in the KVM hypervisor.

Creating a management bridge that is attached to physical interface net device eth1 in the server:

```
sudo brctl addbr mgmt_br0
sudo brctl addif mgmt_br0 eth1
sudo ifconfig eth1 0.0.0.0 up
sudo ifconfig mgmt_br0 <ip> netmask 255.255.255.0 up
sudo ifconfig eth1 up
sudo route add default <gateway ip> mgmt_br0
```

Creating a tap that is used for the -netdev parameter in the QEMU command invocation:

```
sudo openvpn --mktun --dev tap_net1
sudo brctl addif mgmt_br0 tap_net1
sudo ifconfig tap_net1 up
```

Deploying NX-OSv 9000 on VirtualBox

NX-OSv 9000 deployment on VirtualBox uses one of two forms of distributed virtual artifacts: VMDK or Pre-packaged Box. The simplest method is to use Pre-packaged Box along with Vagrant software. However, the box is created for simple standalone VM deployment with very minimal configuration. This procedure is covered in Deploying NX-OSv 9000 on VirtualBox with Vagrant Using a Pre-Packaged Box, on page 23.

Some basic steps and concepts are shown here to create a virtual machine similar to other kinds of VM guests. These instructions are generally for Mac users, but slight differences are highlighted for Window users.

Creating a VM in a VirtualBox Environment Using VMDK

Before You Begin

Hardware

- More than 12 GB of memory.
- More than 4 cores of CPU.

Software

- Mac OS X
  - Install VirtualBox
  - Install Vagrant
  - Install socat for serial console:

```
Homebrew:
  ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
socat:
  brew install socat
```

- Windows
  - Install VirtualBox
  - Install Vagrant
  - Install PuTTY for serial console. (For more information, see www.putty.org.)
  - Install Git to access tools, such as SSH. (For more information, see www.git-for-windows.github.io.)

DETAILED STEPS

	Command or Action	Purpose
Step 1	Copy VMDK to your local hard drive. Launch VirtualBox and click the New icon.	
Step 2	Enter the Name as n9kv, select the Type as Linux, and select the Version as Other Linux (64-bit).	
Step 3	Enter the Memory size (RAM) as 8192 MB.	
Step 4	Select **Use an existing virtual hard drive file**. Click the folder icon (bottom right side) and select the VMDK created in step 1.	
Step 5	Select your VM and click **Settings**.	

	Command or Action	Purpose
Step 6	In the Attributes of Settings, highlight the IDE controller. Select **Add SATA controller** and add the VMDK file. Click **OK** to save the settings.	**Note** Adding the SATA controller enables VirtualBox to boot when the system image size exceeds 512 MB.
Step 7	Click the System icon. • On the Motherboard tab: ◦ Check the Enable EFI checkbox. ◦ Ensure the Modify Boot order so that hard disk is first. • On the Processor tab: ◦ Ensure the Processor is 2. • On the Accelerator tab: ◦ Ensure that VT-x/AMD-V is checked.	
Step 8	Click the Audio icon. Uncheck the Enable Audio checkbox.	
Step 9	Click the Network icon. • On the Adapter 1 tab: ◦ Check the Enable Network Adapter checkbox. ◦ Ensure the Port Forwarding configuration is: ◦ Name = ssh ◦ Host Port = 2222 ◦ Guest Port = 22	
Step 10	Click the Ports icon. • On the Serial Ports tab/Port 1 tab: ◦ Check the Enable Serial Port checkbox. ◦ Ensure that Port Mode is Host Pipe. ◦ Check Create Pipe (or uncheck Connect to existing pipe/socket). ◦ Ensure Port/File Path is /tmp/test.	**Note** For Windows, on the Serial Ports tab/Port 1 tab: • Ensure Path/Address is \\.\pipe\COM1.
Step 11	Click the Ports icon. • On the USB tab:	

	Command or Action	Purpose
	◦ Uncheck the Enable USB 2.0 (EHCI) controller checkbox.	
Step 12	Click the Start button.	Starts the VM. **Note** Ensure that you have the proper serial console setup in your Mac or Windows environment. For more information, see Setup Serial Console on VirtualBox Environment, on page 21.

Setup Serial Console on VirtualBox Environment

Mac OS X

1. Use the 'socat' utility to connect to the serial port (console access) of the VM.
   a. Install Homebrew and then use homebrew to install 'socat'.

```
Homebrew:
  ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
socat:
  brew install socat
```

   b. Run the socat command from an xterm.

```
socat unix-connect:/tmp/test stdin
```

2. Go through the NXOS boot process.
   a. Answer 'no' for the secure password.
   b. Set admin password to 'admin'.
3. Configure the router.

```
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# interface mgmt0
switch(config-if)# ip address 10.0.2.15/24  <--- NOTE: can use "ip address dhcp" here
instead
switch(config-if)# no shut
switch(config-if)# end
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# username vagrant password vagrant role network-admin
switch(config)# username vagrant shell bash
switch(config)# boot nxos bootflash:nxos.7.0.3.I2.2d.bin  <--- Note: use correct image
name from "dir" command output
switch(config)# copy r s
[########################################] 100%
```

```
Copy complete.
switch(config)#
```

Windows

1. Open PuTTY.
2. Open "Basic options for your PuTTY session".
   a. Specify "\\.\pipe\COM1" for Serial line path.
   b. Select "Serial" for "Connection type".
   c. Select "9600" for "Speed".
3. Go through the NXOS boot process.
   a. Answer 'no' for the secure password.
   b. Set admin password to 'admin'.
4. Configure the router.

```
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# interface mgmt0
switch(config-if)# ip address 10.0.2.15/24  <--- NOTE: can use "ip address dhcp" here
instead
switch(config-if)# no shut
switch(config-if)# end
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# username vagrant password vagrant role network-admin
switch(config)# username vagrant shell bash
switch(config)# boot nxos bootflash:nxos.7.0.3.I2.2d.bin  <--- Note: use correct image
name from "dir" command output
switch(config)# copy r s
[########################################] 100%
Copy complete.
switch(config)#
```

Set Up SSH Passwordless Connection to VM

Navigate to login screen and login as vagrant (password: vagrant). Place the vagrant insecure key into the 'authorized_keys' file.

```
switch(config)# exit
switch# exit

User Access Verification
switch login: vagrant
Password:
-bash-4.2$
-bash-4.2$ pwd
/var/home/vagrant
-bash-4.2$ cd .ssh
-bash-4.2$ pwd
/var/home/vagrant/.ssh
-bash-4.2$ echo
"ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA6NF8iallvQVp22WDkTkyrtvp9eWW6A8YVr+kz4Tj
GYe7gHzIw+niNltGEFHzD8+v1I2YJ6oXevct1YeS0o9HZyN1Q9qgCgzUFtdOKLv6IedplqoPkcmF0
aYet2PkEDo3MlTBckFXPITAMzF8dJSIFo9D8HfdOV0IAdx4O7PtixWKn5y2hMNG0zQPyUecp4pzC
6kivAIhyfHilFR61RGL+GPXQ2MWZWFYbAGjyiYJnAmCP3NOTd0jMZEnDkbUvxhMmBYSdETk1rRgm+
R4LOzFUGaHqHDLKLX+FIPKcF96hrucXzcWyLbIbEgE98OHlnVYCzRdK8jlqm8tehUc9c9WhQ
== vagrant insecure public key" > authorized_keys
```

```
-bash-4.2$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA6NF8iallvQVp22WDkTkyrtvp9eWW6A8YVr+kz4TjG
Ye7gHzIw+niNltGEFHzD8+v1I2YJ6oXevct1YeS0o9HZyN1Q9qgCgzUFtdOKLv6IedplqoPkcmF0a
Yet2PkEDo3MlTBckFXPITAMzF8dJSIFo9D8HfdOV0IAdx4O7PtixWKn5y2hMNG0zQPyUecp4pzC6k
ivAIhyfHilFR61RGL+GPXQ2MWZWFYbAGjyiYJnAmCP3NOTd0jMZEnDkbUvxhMmBYSdETk1rRgm+R4
LOzFUGaHqHDLKLX+FIPKcF96hrucXzcWyLbIbEgE98OHlnVYCzRdK8jlqm8tehUc9c9WhQ
== vagrant insecure public key
-bash-4.2$
```

Deploying NX-OSv 9000 on VirtualBox with Vagrant Using a Pre-Packaged Box

Step 1 Open a terminal in your Mac or PC (GitBash) and make a directory (for example, box-test).

Step 2 Download a released image to this directory (for example, nxosv-final.7.0.3.I2.2d.box).

Step 3 Execute "vagrant init".

Step 4 Execute "tar xvd nxosv-final.7.0.3.I2.2d.box nxosv_config.iso" to unpack the NX-OSv switch configuration file. (The nxosv_config.iso file packaged with this .box file is for demonstration purposes. You can customize your own configuration based on CVAC/bootstrap procedures to make your own .iso configuration file.)

Step 5 Execute "vagrant box add box-test nxosv-final.7.0.3.I2.2d.box".

Step 6 Modify the ./Vagrant file to customize all VM configurations. See Vagrant Configuration Examples.

Step 7 Bring up the VM using the "vagrant up" command in the current directory.

Step 8 Wait for a few minutes to let the bootup finish. Then proceed to the next step.

Step 9 Execute "vagrant ssh" to access the NX-OSv bash shell and enter "vagrant" for the password. If "vagrant ssh" fails, go to step 12 (serial console window) to determine the reason for the failure.

Step 10 Execute "su admin" to access the CLI and enter "admin" or the password.

Step 11 Execute the boot image command: "config t; boot nxos bootflash:nxos.7.0.3.I2.2d.bin".

Step 12 Save the configuration using "copy r s".

Step 13 You can monitor the booting process from the serial console using "socat UNIX-CONNECT:/tmp/test STDIN" in another terminal. For Windows PC users, use putty to connect to the serial console through the "\\.\pipe\COM1" name pipe.

Step 14 Shut down the VM using "vagrant halt -f".

Vagrant Configuration Examples

Mac Users

```
# -*- mode: ruby -*-

# vi: set ft=ruby :


# All Vagrant configuration is done below. The "2" in Vagrant.configure

# configures the configuration version (we support older styles for

# backwards compatibility). Please don't change it unless you know what

# you're doing.

Vagrant.configure(2) do |config|

  # The most common configuration options are documented and commented below.
```

```
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://atlas.hashicorp.com/search.
  # config.vm.base_mac = "0800276CEEAA"
  config.vm.define "n9kv1" do |n9kv1|
        n9kv1.vm.box = "box-test"
        n9kv1.ssh.insert_key = false
        n9kv1.vm.boot_timeout = 180
        n9kv1.vm.synced_folder '.', '/vagrant', disabled: true
        n9kv1.vm.network "private_network", ip: "192.168.1.2", auto_config: false,
virtualbox__intnet: "nxosv_network1"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network2"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network3"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network4"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network5"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network6"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network7"
        n9kv1.vm.provider :virtualbox do |vb|
                vb.customize ['modifyvm',:id,'--nicpromisc2','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc3','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc4','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc5','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc6','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc7','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc8','allow-all']
                vb.customize "pre-boot", [
                        "storageattach", :id,

                        "--storagectl", "SATA",
                        "--port", "1",
                        "--device", "0",
```

```
                        "--type", "dvddrive",
                        "--medium", "./nxosv_config.iso",
                ]
        end
  end
end
```

Windows PC Users

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure(2) do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://atlas.hashicorp.com/search.
  # config.vm.base_mac = "0800276CEEAA"
  config.vm.define "n9kv1" do |n9kv1|
        n9kv1.vm.box = "box-test"
        n9kv1.ssh.insert_key = false
        n9kv1.vm.boot_timeout = 180
        n9kv1.vm.synced_folder '.', '/vagrant', disabled: true
        n9kv1.vm.network "private_network", ip: "192.168.1.2", auto_config: false,
virtualbox__intnet: "nxosv_network1"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network2"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network3"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network4"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network5"
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network6"
```

```
        n9kv1.vm.network "private_network", auto_config: false, virtualbox__intnet:
"nxosv_network7"
        n9kv1.vm.provider :virtualbox do |vb|
                vb.customize ['modifyvm',:id,'--uartmode1','server','\\\\.\pipe\COM1']
                vb.customize ['modifyvm',:id,'--nicpromisc2','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc3','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc4','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc5','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc6','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc7','allow-all']
                vb.customize ['modifyvm',:id,'--nicpromisc8','allow-all']
                vb.customize "pre-boot", [
                        "storageattach", :id,

                        "--storagectl", "SATA",
                        "--port", "1",
                        "--device", "0",
                        "--type", "dvddrive",
                        "--medium", "./nxosv_config.iso",
                ]
        end
  end
end
```

Deleting the VM

Step 1 Shut down the VM.

```
nxosv-user@fe-ucs-dt13:~/n9kv/box-test$ vagrant halt --force box-test ==> box-test:
Forcing shutdown of VM...
nxosv-user@fe-ucs-dt13:~/n9kv/box-test$
```

Step 2 Delete the VM from the system.

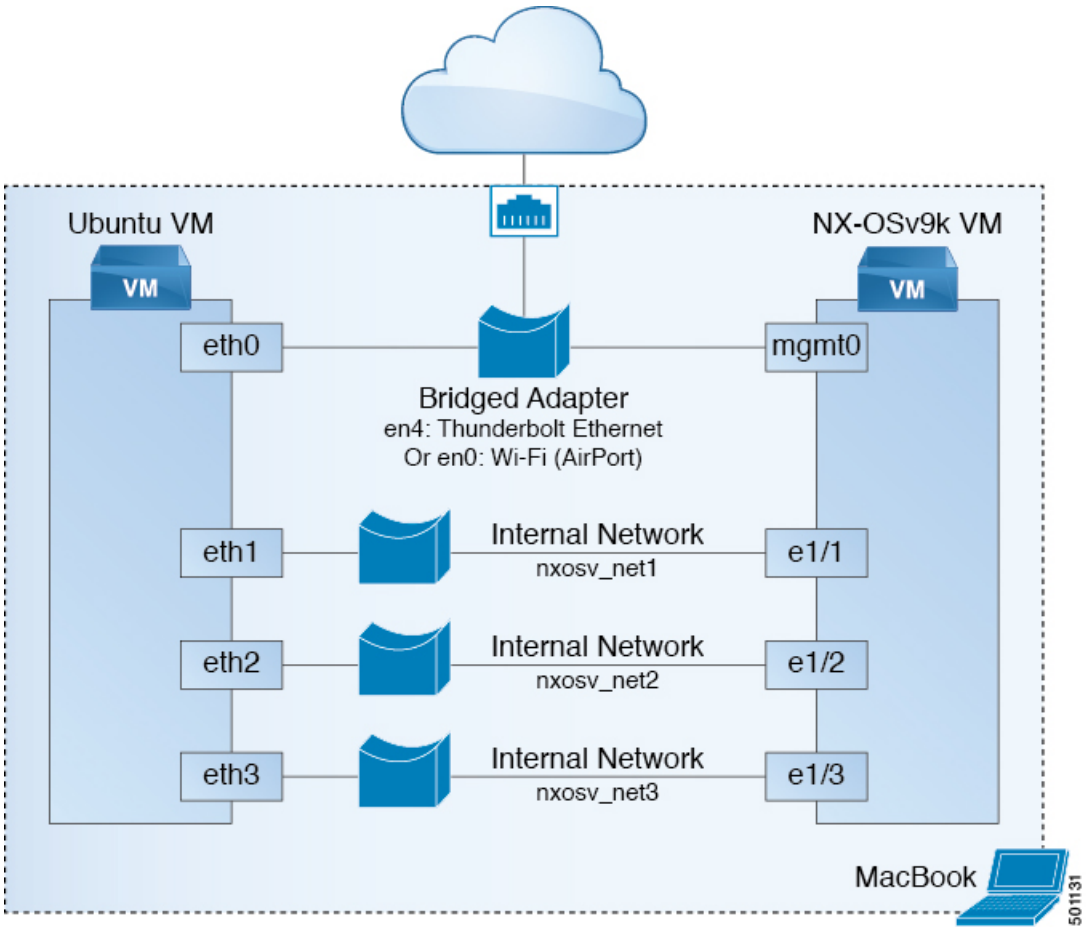
```
nxosv-user@fe-ucs-dt13:~/n9kv/box-test$ vagrant destroy box-test
    box-test: Are you sure you want to destroy the 'box-test' VM? [y/N] y
==> box-test: Destroying VM and associated drives...
nxosv-user@fe-ucs-dt13:~/n9kv/box-test$
```

Network Topology Examples

A key advantage of NX-OSv 9000 is that you can set up a quick networking topology without hardware or complicated cabling tasks to obtain a look and feel about a Cisco Nexus 9000 switch platform.

For example, you can quickly set up a two node system with a server connecting to a NX-OSv 9000 virtual machine on laptop. A more complex system can also be setup with a large resource server to do a multiple node simulation. With the topology, you can do tooling and automation in a simulated network that could be applied in a real customer network environment. The following examples show how to interconnect VMs on a laptop or UCS servers.

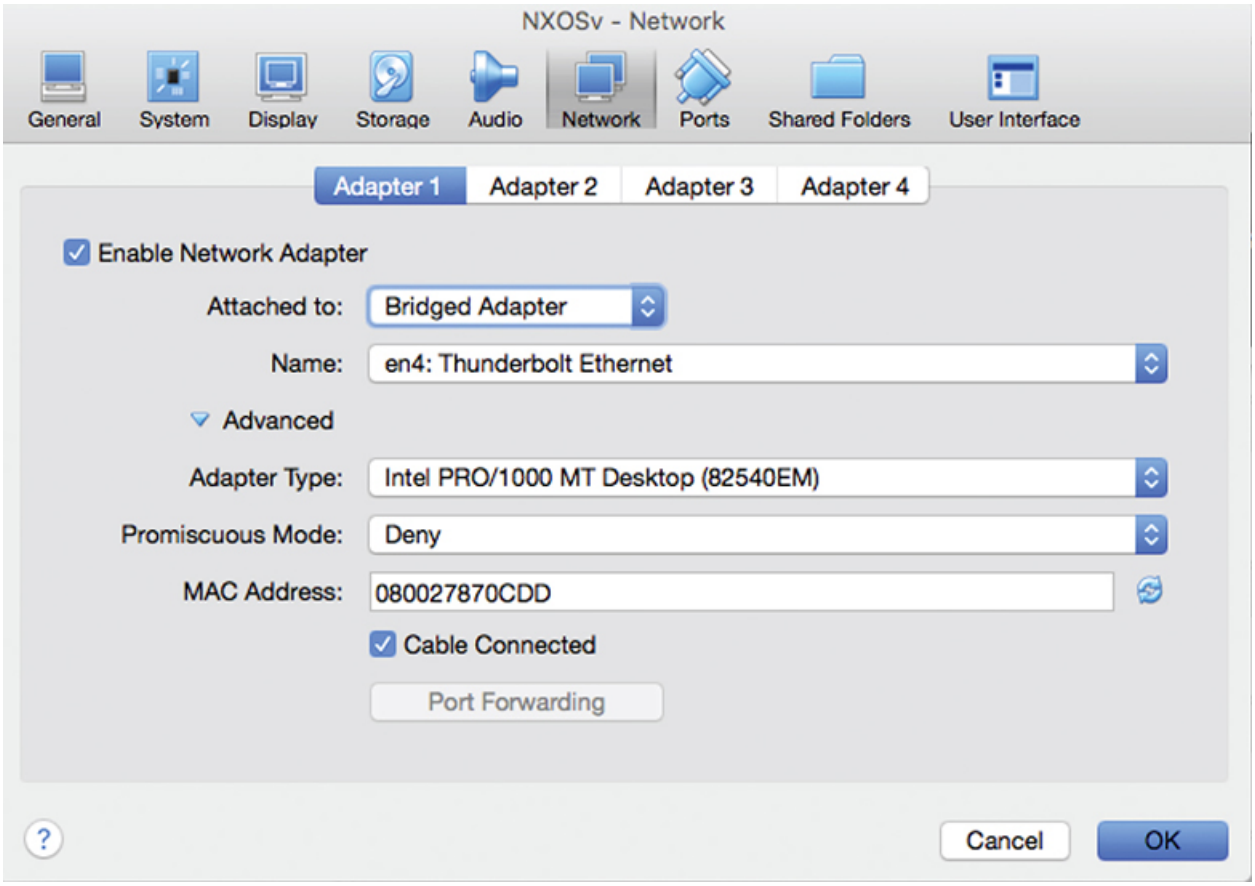
VirtualBox Topology on a Laptop

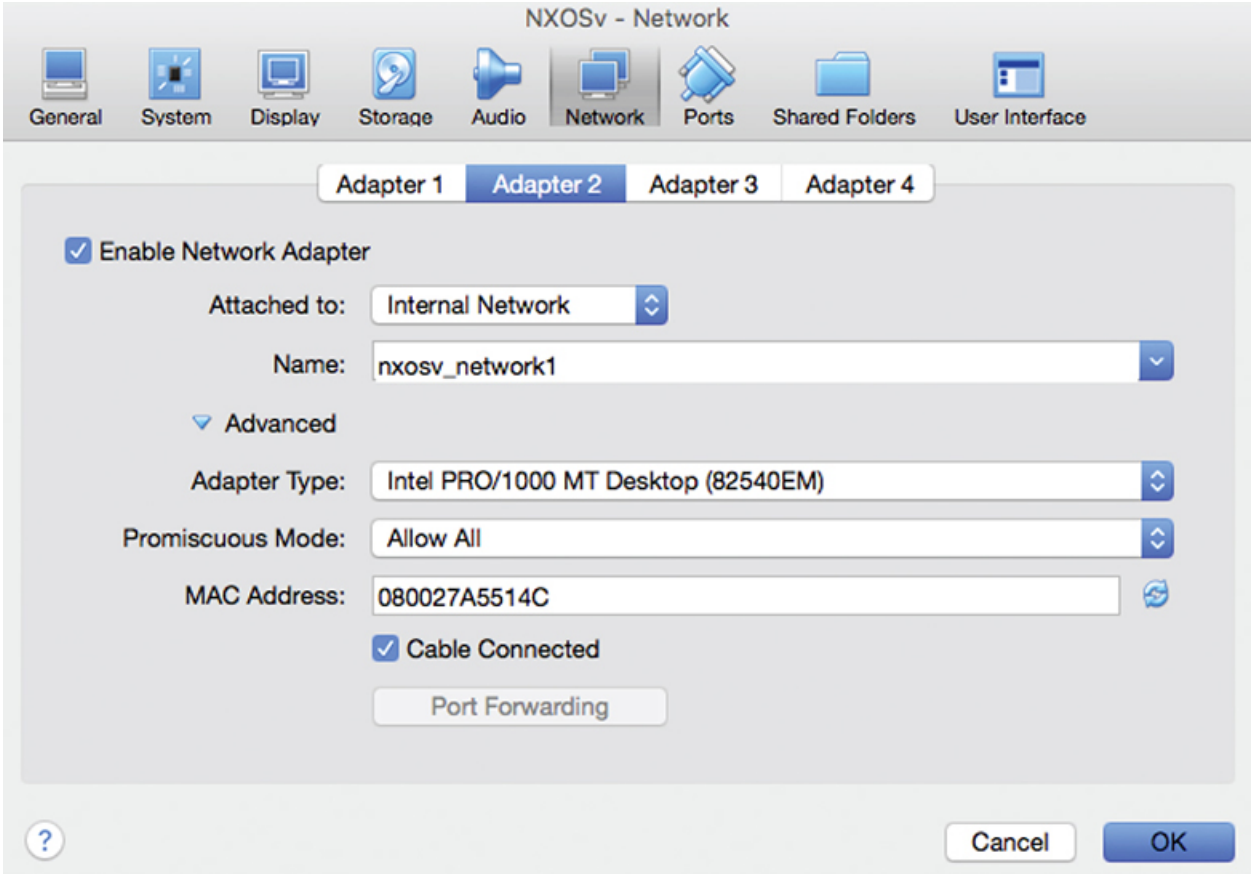


An example diagram above is a typical configuration with NX-OSv 9000 and Ubuntu VM two node system. In this case, Both Ubuntu VM and NX-OSv 9000 would obtain IPs statically or dynamically visa DHCP protocol reachable from cloud. Similarly, both Ubuntu and NX-OSv 9000 can be managed through management network. Ubuntu VM can send/receive packets to NX-OSv 9000 through NX-OSv 9000 data ports, eth1/1, eth1/2, and eth1/3, or ... e1/9.

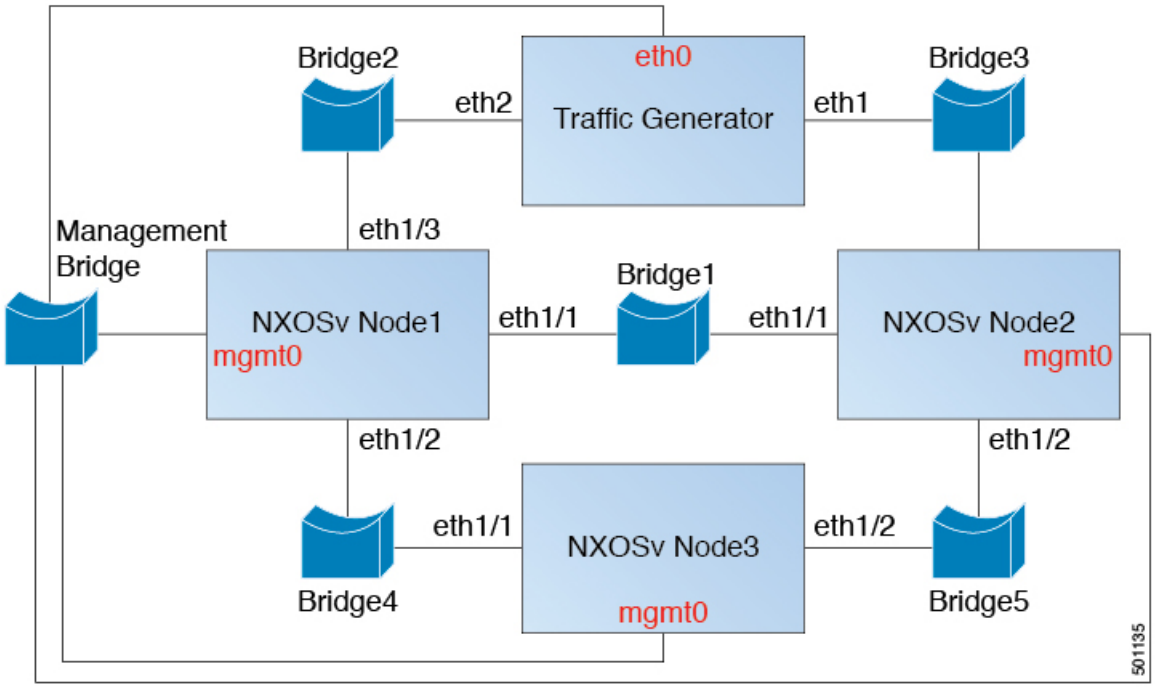
Key to Setup:

- Bridge or NAT to Laptop physical ethernet port for management connectivity
- Internal Network for data ports between VMs, change "Promiscuous Mode" to "Allow All"




NXOSv - Network
General
System
Display
Storage
Audio
Network
Ports
Shared Folders
User Interface
Adapter 1
Adapter 2
Adapter 3
Adapter 4
Enable Network Adapter
Attached to:
Internal Network
Name:
nxosv_network1
Advanced
Adapter Type:
Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode:
Allow All
MAC Address:
080027A5514C
Cable Connected
Port Forwarding
Cancel
OK

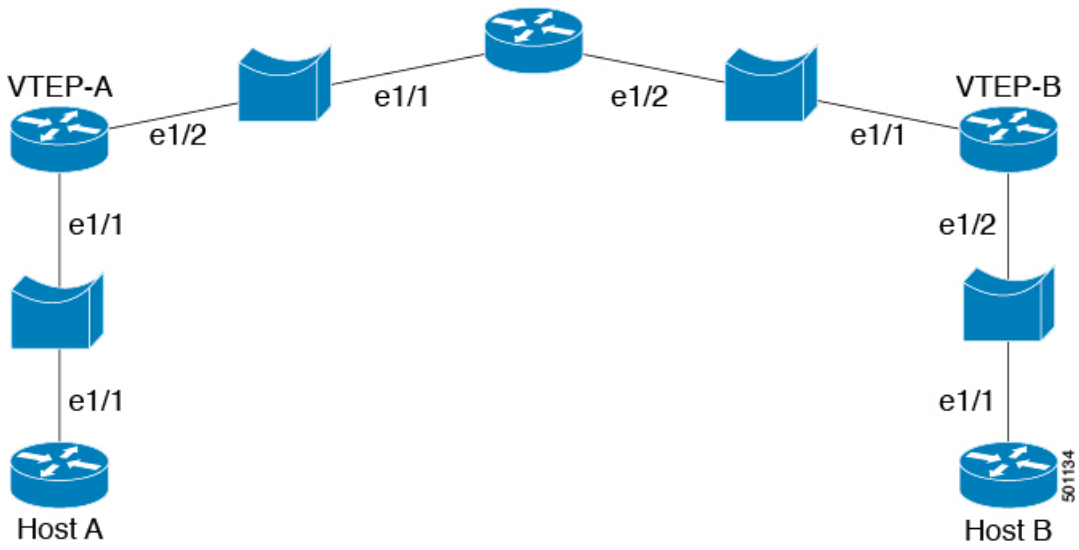
Three Node Topology with Traffic Generator



The nodes in the above diagram are instantiated using the hypervisor specific machine definitions. For networking, each data port interface pair needs to be connected to unique bridge/vSwitch. All the management ports of the NX-OSv 9000 (mgmt0) need to be connected to the management bridge and provided a unique IP address, which will enable access to these devices from an external network.

Each data port interface pair that needs to be interconnected should be mapped to the same Bridge/vSwitch. Similar to VirtualBox topology, vSwitch/Bridge must have "Promiscuous Mode" set to "Accept" and "Vlan ID" to "All" for networking to work between NX-OSv 9000 nodes. Please read "Troubleshooting" section for hypervisor specific handling for data port communication.

Five Nodes VXLAN Topology



This topology can simulate basic vxlan functionality on NX-OSv 9000 platform. Similar bridge/vSwitch setup should be done as shown in other topology examples.