

Kernel Stack

- About Kernel Stack, on page 1
- Guidelines and Limitations, on page 1
- Changing the Port Range, on page 2
- About VXLAN with kstack, on page 3
- Netdevice Property Changes, on page 4

About Kernel Stack

Kernel Stack (kstack) uses well known Linux APIs to manage the routes and front panel ports.

Open Containers, like the Guest Shell, are Linux environments that are decoupled from the host software. You can install or modify software within that environment without impacting the host software packages.

Kernel Stack has the following features:

Guidelines and Limitations

- Guest shell, Docker containers, and the host Bash Shell use Kernel Stack (kstack).
- The Guest Shell and the host Bash Shell start in the default network namespace. Docker containers start in the management network namespace by default.
  - Other network namespaces may be accessed by using the **setns** system call
  - The **nsenter** and **ip netns exec** utilities can be used to execute within the context of a different network namespace.
- The interface state may be read from `/proc/net/dev` or retrieved using other typical Linux utilities such as **ip**, **ifconfig**, or **netstat**. The counters are for packets that have initiated or terminated on the switch.
- **ethtool –S** may be used to get extended statistics from the net devices, which includes packets that are switched through the interface.
- Packet capture applications like **tcpdump** may be run to capture packets that are initiated from or terminated on the switch.

- There is no support for networking state changes (interface creation or deletion, IP address configuration, MTU change, and so on) from the Guest Shell.
- IPv4 and IPv6 are supported.
- Raw PF_PACKET is supported.
- Only on stack (Netstack or kstack) at a time can use well-known ports (0-15000), regardless of the network namespace.
- There is no IP connectivity between applications using Nestack and applications running kstack on the same switch. This limitation holds true regardless of whether the kstack applications are being run from the host Bash Shell or within a container.
- Applications within the Guest Shell are not allowed to send packets directly over an Ethernet out-of-band channel (EOBC) interface to communicate with the line cards or standby Sup.
- The management interface (mgmt0) is represented as eth1 in the kernel netdevices.
- Use of the VXLAN overlay interface (NVE x) is not supported for applications utilizing the kernel stack. NX-OS features, including CLI commands, are able to use this interface via netstack.

  For more information about the NVE interface, see the Cisco Nexus 9000 Series NX-OS VXLAN Configuration Guide.

Changing the Port Range

Netstack and kstack divide the port range between them. The default port ranges are as follows:

- Kstack—15001 to 58000
- Netstack—58001 to 65535

Note Within this range 63536 to 65535 are reserved for NAT.

Note The ports configured with **nxapi use-vrf management** uses kstack and are accessible.

SUMMARY STEPS

1. [**no**] **sockets local-port-range** *start-port end-port*

DETAILED STEPS

Procedure

	Command or Action	Purpose
Step 1	[**no**] **sockets local-port-range** *start-port end-port*	This command modifies the port range for kstack. This command does not modify the Netstack range.

Example

The following example sets the kstack port range:

```
switch# sockets local-port-range 15001 25000
```

What to do next

After you have entered the command, be aware of the following issues:

- Reload the switch after entering the command.
- Leave a minimum of 7000 ports unallocated which are used by Netstack.
- Specify the *start-port* as 15001 or the *end-port* as 65535 to avoid holes in the port range.

About VXLAN with kstack

Starting with NX-OS 9.2(1), VXLAN EVPN is supported with kstack to be leveraged by third-party applications. This functionality is supported on the Cisco Nexus 9000 ToR switches.

Setting Up VXLAN for kstack

No additional configuration is required to make the interfaces or network namespaces for VXLAN EVPN accessible to the third-party applications. The VXLAN EVPN routes are programmed automatically in the kernel based on the NX-OS VXLAN EVPN configuration. For more information, see the "Configuring VXLAN BGP EVPN" chapter in the *Cisco Nexus 9000 Series NX-OS VXLAN Configuration Guide*.

Troubleshooting VXLAN with kstack

To troubleshoot VXLAN issues, enter the following command to list several critical pieces of information to be collected:

```
switch(config)# show tech-support kstack
```

- Run the **ip route show** command:

```
root@switch(config)# run bash sudo su-
root@switch# ip netns exec evpn-tenant-kk1 ip route show
```

Output similar to the following appears:

```
10.160.1.0/24 dev Vlan1601 proto kernel scope link src 10.160.1.254
10.160.1.1 dev veth1-3 proto static scope link metric 51
10.160.2.0/24 dev Vlan1602 proto kernel scope link src 10.160.2.253
127.250.250.1 dev veth1-3 proto static scope link metric 51
```

Verify that all EVPN routes for the corresponding VRF are present in the kernel.

- Run the **ip neigh show** command:

```
root@switch(config)# run bash sudo su-
root@switch# ip netns exec evpn-tenant-kk1 ip neigh show
```

Output similar to the following appears:

```
10.160.1.1 dev veth1-3 lladdr 0c:75:bd:07:b4:33 PERMANENT
127.250.250.1 devveth1-3 lladdr0c:75:bd:07:b4:33 PERMANENT
```

Netdevice Property Changes

Starting with the NX-OS 9.2(2) release, netdevices representing the front channel port interfaces are always in the ADMIN UP state. The final, effective state is determined by the link carrier state.

The following example shows the following interfaces in NX-OS, where eth1/17 is shown as **up** and eth1/1 is shown as **down**:

```
root@kstack-switch# sh int ethernet 1/17 brief
Eth1/17       --      eth  routed up      none                       1000(D) --


root@kstack-switch# sh int ethernet 1/1 brief
Eth1/1        --      eth  routed down    Link not connected         auto(D) --
```

The following example shows these same interfaces, but this time as shown in the Bash shell using the **ip link show** command:

```
bash-4.3# ip link show Eth1-17
49: Eth1-17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT group default qlen 100
    link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff


bash-4.3# ip link show Eth1-1
33: Eth1-1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode
DEFAULT group default qlen 100
    link/ether 00:42:68:58:f8:eb brd ff:ff:ff:ff:ff:ff
```

In this example, Eth1-1 is shown as being **UP**, but is shown as **NO-CARRIER** and **state DOWN**.

The following example shows these same interfaces, but this time as shown in the Bash shell using the **ifconfig** command:

```
bash-4.3# ifconfig Eth1-17
Eth1-17   Link encap:Ethernet  HWaddr 00:42:68:58:f8:eb
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7388 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:1869164 (1.7 MiB)


bash-4.3# ifconfig Eth1-1
Eth1-1    Link encap:Ethernet  HWaddr 00:42:68:58:f8:eb
          inet addr:99.1.1.1  Bcast:99.1.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

The output from the **ifconfig** command provides different information, where the **RUNNING** keyword is used to represent the final state. By default, all netdevices show the keyword **UP**, which represents the ADMIN state of the netdevice in the kernel.

Following are the changes that are part of the NX-OS 9.2(2) release:

- **IPv4 address on netdevices** — Before the NX-OS 9.2(2) release, the IPv4 address would be plumbed to the netdevice in the kernel even when the corresponding interface in NX-OS was in the **DOWN** state. Starting with the NX-OS 9.2(2) release, the IPv4 address are plumbed to the kernel space only when the interface is in the **UP** state. Once plumbed, the IPv4 address continues to stay with the netdevice in the kernel even if the interface goes **DOWN**. It will be removed only after you have entered the following CLI command to explicitly remove the IP address from the NX-OS interface:

  ```
  Interface Eth1/1
              no ip address IP-address
  ```

- **IPv6 address on netdevices** — Before the NX-OS 9.2(2) release, the IPv6 address would get flushed from the netdevices in the kernel when the interface was **DOWN**. Starting with the NX-OS 9.2(2) release, the netdevices are always in the admin **UP** state, so the IPv6 addresses will not get flushed from the kernel when the interface goes down.