Container Network - Linux Networking Basics

NETWORK NAMESPACE INSPECTING

Linux Network Stack
network-stack

In Linux networking, within the Linux network stack, routes define traffic paths, iptables configures packet filtering, lo is a local loopback interface for testing, and eth0 is the primary Ethernet interface for external connections. Let's inspect the network stack, in short.

Network interfaces allow us to establish communication between a network and a device.

```
ip link list
```

lo is the loopback interface, allowing local network communication within a device without external network involvement. Verify the loopback interface is up

```
ifconfig lo
```

A route in networking specifies the path for network traffic from source to destination. View the routing table:

```
ip route show
```

iptables is a user-space utility for configuring packet filter rules in the Linux kernel's Netfilter framework. View iptables rules:

```
iptables -L
```

Create Custom Network Namespace
Let's create a custom namespace using ip netns add utiliy.

```
sudo ip netns add poridhi
sudo ip netns list
```

Now, entering a network namespace in Linux:

```
ip netns exec
```

It is part of the iproute2 package and is often used for managing network namespaces.

```
sudo ip netns exec poridhi bash
```

nsenter

The nsenter utility is commonly used to enter into namespaces in Linux, including network namespaces.

```
sudo nsenter --net=/var/run/netns/poridhi bash
```

Now, check the network interfaces inside the new ns.

ip link show
Let's check for iptable rules for custom ns.

ifconfig lo

explain these for a simple child in a very easiest manner in details with basics also correct any error

Sure! Let's break everything down in the easiest way possible. Imagine your computer is like a big city, and different parts of the city (apps and services) need to talk to each other. The **Linux Network Stack** is like the system of roads, traffic lights, and highways that let information move from one place to another.

Understanding the Basics of Networking in Linux

Think of a **network interface** like a door to your house. It lets information (data) come in and go out.

1. Listing Network Interfaces

To see all the doors (network interfaces) in your computer, use:

```bash
ip link list
```

This will show you all the available network interfaces.

- **lo (Loopback Interface)**: This is a special door inside your own house. It lets programs on your computer talk to each other **without using the internet**.
- **eth0 (Ethernet Interface)**: This is like the main gate of your house. It lets you connect to the outside world (like the internet).

To check if the **loopback interface** (lo) is working, use:

```bash
ifconfig lo
```

If it's up, it means your computer can communicate with itself.

---

2. Viewing the Routes (Roads for Data)

In a city, roads guide where cars go. In a computer network, **routes** guide where data should go.

To see the list of roads (routes), use:

```bash
ip route show
```

This tells the computer how to send and receive messages.

---

3. Understanding iptables (Security Guards for Traffic)

Imagine security guards at checkpoints controlling what enters and leaves the city. **iptables** is a tool that acts like these security guards, deciding which data packets can pass through.

To check the security rules, use:

```
bash

iptables -L
```

This will list all the rules currently in place.

Creating a Separate "City" (Network Namespace)

A **network namespace** is like creating a new city inside your computer. It will have its own separate roads, doors, and security guards.

To create a new **network namespace**, use:

```
bash

sudo ip netns add poridhi
```

Now, to check if it was created, use:

```
bash

sudo ip netns list
```

It will show a list of all network namespaces, including "poridhi".

Entering the New Namespace (Visiting the New City)

Once we create a new network namespace, we need to **enter** it to check things inside.

Method 1: Using `` `ip netns exec` ``

```
bash

sudo ip netns exec poridhi bash
```

This command lets us "step inside" the new city (namespace).

Method 2: Using `` `nsenter` ``

```
bash

sudo nsenter --net=/var/run/netns/poridhi bash
```

This is another way to enter the new city.

Checking the Network Inside the New Namespace

Now that we're inside the new network namespace, let’s check what network interfaces exist.

```bash
ip link show
```

Since it's a new network space, it may not have `eth0` yet.

To check if the loopback interface is working inside the new namespace:

```bash
ifconfig lo
```

If it's not up, you may need to turn it on:

```bash
ip link set lo up
```

Summary

1. `ip link list` → Shows all network interfaces (doors).
2. `ifconfig lo` → Checks if the computer can talk to itself.
3. `ip route show` → Shows data paths (roads).
4. `iptables -L` → Shows security rules.
5. `ip netns add poridhi` → Creates a separate network space.
6. `sudo ip netns list` → Lists all network spaces.
7. `sudo ip netns exec poridhi bash` → Enters the new network space.
8. `ip link show` → Checks network interfaces inside the new space.
9. `ifconfig lo` → Checks loopback inside the namespace.

This is like setting up a small city inside your computer, separate from the main one. 🚀

explain these terminals:

```
brd 00:00:00:00:00:00
2: bond0: <BROADCAST,MULTICAST,MASTE
R> mtu 1500 qdisc noqueue state DOWN
 mode DEFAULT group default qlen 100
0
```

```
    link/ether d2:25:4b:d2:1e:34 brd
 ff:ff:ff:ff:ff:ff
3: dummy0: <BROADCAST,NOARP> mtu 150
0 qdisc noop state DOWN mode DEFAULT
 group default qlen 1000
    link/ether 6e:6f:ec:6d:af:a8 brd
 ff:ff:ff:ff:ff:ff
4: eth0: <BROADCAST,MULTICAST,UP,LOW
ER_UP> mtu 1500 qdisc pfifo_fast sta
te UP mode DEFAULT group default qle
n 1000
    link/ether b6:b1:28:ec:3e:1f brd
 ff:ff:ff:ff:ff:ff
5: docker0: <NO-CARRIER,BROADCAST,MU
LTICAST,UP> mtu 1500 qdisc noqueue s
tate DOWN mode DEFAULT group default

    link/ether 02:42:b0:d6:bc:97 brd
 ff:ff:ff:ff:ff:ff
root@50cd87437a310287:~# ifconfig lo
lo: flags=73<UP,LOOPBACK,RUNNING>  m
tu 65536
        inet 127.0.0.1  netmask 255.
0.0.0
        inet6 ::1  prefixlen 128  sc
opeid 0x10<host>
        loop  txqueuelen 1000  (Loca
l Loopback)
        RX packets 0  bytes 0 (0.0 B
)
        RX errors 0  dropped 0  over
runs 0  frame 0
        TX packets 0  bytes 0 (0.0 B
)
        TX errors 0  dropped 0 overr
uns 0  carrier 0  collisions 0
root@50cd87437a310287:~# ip route show
default via 10.62.0.1 dev eth0
10.62.0.0/16 dev eth0 proto kernel scope link src 10.62.18.251
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
root@50cd87437a310287:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
Chain FORWARD (policy DROP)
target     prot opt source               destination
DOCKER-USER  all  --  anywhere             anywhere
DOCKER-ISOLATION-STAGE-1  all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED,
ESTABLISHED
DOCKER     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Chain DOCKER (1 references)
target     prot opt source               destination
Chain DOCKER-ISOLATION-STAGE-1 (1 references)
target     prot opt source               destination
DOCKER-ISOLATION-STAGE-2  all  --  anywhere             anywhere
RETURN     all  --  anywhere             anywhere
Chain DOCKER-ISOLATION-STAGE-2 (1 references)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere
RETURN     all  --  anywhere             anywhere
Chain DOCKER-USER (1 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere
root@50cd87437a310287:~#
```

Let's break down the terminal outputs into simple explanations.

---

1. Understanding the Network Interfaces

When you run `ip link list`, it shows the network interfaces on your system. Here's what each one means:

brd 00:00:00:00:00:00

- `brd` stands for **broadcast address**.
- `00:00:00:00:00:00` means it's not set (default).
- **Broadcasting** is when a network message is sent to all devices in a network.

2: bond0

```bash
2: bond0: <BROADCAST,MULTICAST,MASTER> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether d2:25:4b:d2:1e:34 brd ff:ff:ff:ff:ff:ff
```

- **bond0**: This is a **bonded interface**, meaning it combines multiple network connections for reliability or speed.
- **BROADCAST, MULTICAST, MASTER**:
  - **BROADCAST**: Can send packets to all devices in the network.
  - **MULTICAST**: Can send packets to a group of devices.
  - **MASTER**: This interface manages bonded network connections.
- **MTU 1500**: Maximum packet size is 1500 bytes.
- **State DOWN**: It is currently inactive.

- **MAC Address**: **`` `d2:25:4b:d2:1e:34` ``** (unique hardware identifier).

3: dummy0

```
bash

3: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 6e:6f:ec:6d:af:a8 brd ff:ff:ff:ff:ff:ff
```

- **dummy0**: A **dummy network interface**, often used for testing or software purposes.
- **NOARP**: This interface does **not use ARP** (Address Resolution Protocol).
- **State DOWN**: It is currently inactive.

4: eth0

```
bash

4: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group de
    link/ether b6:b1:28:ec:3e:1f brd ff:ff:ff:ff:ff:ff
```

- **eth0**: This is the main **Ethernet network interface** (wired internet).
- **UP, LOWER_UP**: The network interface is active and connected.
- **pfifo_fast**: A queuing discipline used to manage network traffic.

5: docker0

```
bash

5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT grou
    link/ether 02:42:b0:d6:bc:97 brd ff:ff:ff:ff:ff:ff
```

- **docker0**: This is a virtual network bridge created by **Docker** to connect containers.
- **NO-CARRIER**: No physical network is connected to it.
- **State DOWN**: It is not currently in use.

---

2. Loopback Interface (`` `lo` ``)

```
bash

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
```

- **lo (Loopback Interface)**: A virtual interface used for internal communication **within the same computer**.
- **IP Address 127.0.0.1**: Used to send data to itself (localhost).
- **MTU 65536**: This allows very large packets (much bigger than normal interfaces).

3. Network Routing Table (`ip route show`)

```
bash

default via 10.62.0.1 dev eth0
10.62.0.0/16 dev eth0 proto kernel scope link src 10.62.18.251
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
```

- **default via 10.62.0.1 dev eth0**: All outgoing internet traffic is sent through `eth0` (wired internet).
- **10.62.0.0/16 dev eth0**: The computer is part of a local network (subnet `10.62.0.x`).
- **172.17.0.0/16 dev docker0**: This is Docker's internal network.
    - `linkdown`: Docker network is not active.

---

4. Firewall Rules (`iptables -L`)

This is a list of firewall rules that control network traffic.

Chain INPUT (Incoming Traffic)

```
bash

Chain INPUT (policy ACCEPT)
target     prot opt source               destination
```

- **Policy ACCEPT**: No restrictions for incoming traffic.

Chain FORWARD (Traffic Routing)

```
bash

Chain FORWARD (policy DROP)
```

- **Policy DROP**: By default, the system **blocks** forwarded traffic (useful in routers).
- Other rules:

```
bash

DOCKER-USER  all  --  anywhere             anywhere
DOCKER-ISOLATION-STAGE-1  all  --  anywhere             anywhere
ACCEPT     all  --  anywhere             anywhere             ctstate RELATED, ESTABLISHED
```

- **DOCKER Rules**: Docker has its own firewall rules to isolate container traffic.
- **ACCEPT all RELATED, ESTABLISHED**: Allow existing connections.

Chain OUTPUT (Outgoing Traffic)

```bash
Chain OUTPUT (policy ACCEPT)
```

- The system allows all outgoing traffic by default.

Docker-Specific Chains

```bash
Chain DOCKER-ISOLATION-STAGE-1
Chain DOCKER-ISOLATION-STAGE-2
```

- These rules **isolate** Docker containers so they don't interfere with each other.

Summary

1. `` `ip link list` `` → Lists all network interfaces.
2. **Loopback (`` `lo` ``)** → Used for internal communication (`` `127.0.0.1` ``).
3. **Bonded Interface (`` `bond0` ``)** → Merges multiple connections.
4. **Dummy Interface (`` `dummy0` ``)** → Fake network interface (for testing).
5. **Ethernet Interface (`` `eth0` ``)** → Main internet connection.
6. **Docker Bridge (`` `docker0` ``)** → Connects containers.
7. `` `ip route show` `` → Shows network routes.
8. `` `iptables -L` `` → Shows firewall rules.

This setup is common on Linux machines running Docker or virtualized environments! 🚀

explain these terminal and how to execute command inside a custom namespaces:

```
root@50cd87437a310287:~# ip route show
root@50cd87437a310287:~# sudo ip netns add poridhi
Cannot create namespace file "/run/netns/poridhi": File exists
root@50cd87437a310287:~# sudo ip netns list
poridhi
i bash0cd87437a310287:~# sudo ip netns exec poridhi
root@50cd87437a310287:~# ip link show
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default q
len 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
i bash0cd87437a310287:~# sudo ip netns exec poridhi
i bash0cd87437a310287:~# sudo ip netns exec poridhi
root@50cd87437a310287:~# ifconfig lo
lo: flags=8<LOOPBACK>  mtu 65536
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
root@50cd87437a310287:~#
```

Let's go through your terminal commands step by step and explain everything in simple terms.

---

1. Checking Routing Table

```bash
root@50cd87437a310287:~# ip route show
```

- This command shows the **routing table**, which tells the system where to send network traffic.
- It helps understand how data flows inside and outside your machine.

---

2. Creating a Network Namespace

```bash
root@50cd87437a310287:~# sudo ip netns add poridhi
Cannot create namespace file "/run/netns/poridhi": File exists
```

- `sudo ip netns add poridhi` → This command tries to create a network namespace named `poridhi`.
- The error **"File exists"** means the namespace `poridhi` **already exists**, so you don't need to create it again.

---

3. Listing Available Network Namespaces

```bash
root@50cd87437a310287:~# sudo ip netns list
poridhi
```

- This confirms that `poridhi` already exists.
- It shows a list of all created **network namespaces**.

---

4. Executing a Command Inside the Namespace

```bash
root@50cd87437a310287:~# sudo ip netns exec poridhi
```

- `ip netns exec poridhi` → This command runs a command **inside the namespace** `poridhi`.

- However, in your output, there's no command after **`exec poridhi`**, so nothing happens.
- To enter the namespace properly, you should use:

  ```
  bash

  sudo ip netns exec poridhi bash
  ```

  This will start a new **Bash shell** inside the **`poridhi`** namespace.

---

5. Checking Network Interfaces Inside the Namespace

```
bash

root@50cd87437a310287:~# ip link show
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

- **`ip link show`** lists all network interfaces inside the current namespace.
- Since we haven't created any custom interfaces inside **`poridhi`**, the only interface present is:
  - **lo (Loopback Interface)**: Used for internal communication within the namespace.
  - **State DOWN**: It is currently disabled.
- To enable the loopback interface inside **`poridhi`**, use:

  ```
  bash

  sudo ip netns exec poridhi ip link set lo up
  ```

---

6. Checking the Loopback Interface (`lo`)

```
bash

root@50cd87437a310287:~# ifconfig lo
lo: flags=8<LOOPBACK>  mtu 65536
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        TX packets 0  bytes 0 (0.0 B)
```

- **`ifconfig lo`** shows the status of the **`lo`** (loopback) interface.
- **Flags** **`<LOOPBACK>`** → This confirms it is a loopback interface.
- **MTU 65536** → Allows large packets inside the namespace.
- **RX and TX packets 0** → No data has been sent or received yet.

How to Properly Execute Commands Inside a Namespace

Entering the Namespace with a Bash Shell

```bash
sudo ip netns exec poridhi bash
```

- This will start a new Bash session **inside** the namespace.

Running Commands Directly Inside the Namespace

If you just want to run a single command inside `poridhi`, use:

```bash
sudo ip netns exec poridhi <command>
```

Examples:

- **Check network interfaces inside `poridhi`:**

  ```bash
  sudo ip netns exec poridhi ip link show
  ```

- **Enable the loopback interface inside `poridhi`:**

  ```bash
  sudo ip netns exec poridhi ip link set lo up
  ```

- **Check the IP address inside `poridhi`:**

  ```bash
  sudo ip netns exec poridhi ip addr show
  ```

- **Ping another IP from inside `poridhi`:**

  ```bash
  sudo ip netns exec poridhi ping 8.8.8.8
  ```

Summary

1. **Check network routes:** `ip route show`
2. **Create a namespace:** `sudo ip netns add poridhi`
3. **List namespaces:** `sudo ip netns list`
4. **Enter namespace shell:** `sudo ip netns exec poridhi bash`
5. **Run commands inside namespace:**
    - `sudo ip netns exec poridhi ip link show` → List interfaces

  - ``` `sudo ip netns exec poridhi ip link set lo up` ``` → Enable loopback
  - ``` `sudo ip netns exec poridhi ping 8.8.8.8` ``` → Ping from namespace

Let me know if you need more clarification! 🚀