Chapter 1

Configuring Advanced Networking

1.1 Networking Basics Resumed

1.1.1 Network configuration tools

Terms	Description
ip addr show	Shows address information about all network interfaces.
ip -s link show ens33	Shows statistics about packets but for interface `ens33`. Same as `ip -s link`, but for a specific interface.
ip route	Shows routing information
traceroute / **tracepath**	For analysing a particular route or path.
netstat / **ss**	Analyse ports and services currently listening for incoming connections.

1.1.2 Network Manager

NetworkManager is used to both manage and monitor network settings. While the settings made with the IP tool act directly on the NICs, they're temporary and wiped with every boot or even bringing the interface down and up again. The network manager uses config scripts in `/etc/sysconfig/network-scripts` to store our configs and use them after every boot. The settings can be managed using either `nmcli` or `nmtui`. The former is preferred for scripts while `nmtui` is preferred for manual configs.

nmcli concepts

- A **device** or an **interface** is a network interface, corresponding to the hardware NIC (Network Interface Card).
- A **connection** is a collection of configuration settings for a *device*.
- Multiple connections can exist for the same device, but since they operate on the same settings for the device, only one of them can be active.

- All the connections (and some details) can be shown with the command `nmcli con show`.
- To show all the details for a particular connection, we have to use the command `nmcli con show <interface name>` like `nmcli con show wlo1` (where *wlo1* is the name of the connection).
- To see the connection status for a device, we use `nmcli dev status`. This shows us which devices are connected and which connection they're presently using.
- To see the details of the actual NIC device, we use `nmcli dev show <deviceName>`.

1.1.3 Creating Network Interfaces with nmcli

To add a new connection using `nmcli` that has the name *dhcp* that auto-connects using dynamic IP on interface *eno1*, we use:

```
# nmcli con add con-name "dhcp" type ethernet ifname eno1
```

To add a new connection *static* that uses a static ip that doesn't connect automatically, we use:

```
# nmcli con add con-name "static" type ethernet ifname eno1 autoconnect no ip4
↪  192.168.122.102 gw4 192.168.122.1
```

Now, the available connections can be checked with `nmcli dev status`. The we can connect the *static* connection using `nmcli con up static` and then switch back to the original connection *dhcp* using `nmcli con up dhcp`.

1.1.4 Modifying Network Interfaces using nmcli

To see the details of the *static* connection, we use `nmcli con show static`. Then, to add/modify the DNS server address for that connection, we use the `con mod` keywords, which makes the command:

```
# nmcli con mod "static" ipv4.dns 192.168.122.1
```

Note that the modification requires the `ipv4` keyword instead `ip4`. To define a second IPv4 DNS for the *static* connection, we use the + symbol to denote that a new value for the item should be added and the old value shouldn't be overwritten. The command then becomes:

```
# nmcli con mod "static" +ipv4.dns 8.8.8.8
```

An existing static IP address and gateway can be edited using:

```
# nmcli con mod "static" ipv4.addresses "192.168.100.10/24 192.168.100.1"
```

A secondary IPv4 address can be added using:

```
# nmcli con mod "static" +ipv4.addresses "10.0.0.10/24"
```

Finally, to activate all the above settings, we use: `nmcli con up static`.

1.1.5 Working directly with Configuration Files

All the `nmcli` tool really does while adding or modifying settings is write the changes to the configuration files in `/etc/sysconfig/network-scripts/ifcfg-<interfaceName>`. We may choose to edit them directly if needed. Then, after making the necessary modifications, we ask the NetworkManager service to reload the configuration using `nmcli con reload`.

1.1.6 Managing Hostname and DNS

The hostname is stored in the file `/etc/hostname` and can be edited directly or using the `hostnamectl set-hostname <newHostName>` command. The current hostname can then be viewed using `hostnamectl status`.

The value of the search domain and preferred nameserver (i.e., the one that the NetworkManager uses by default) is auto-pushed from `/etc/sysconfif/network-scripts/ifcfg-<connectionName>` to the file `/etc/resolv.conf`.

1.2 Understanding Routing

Let us consider the following network:

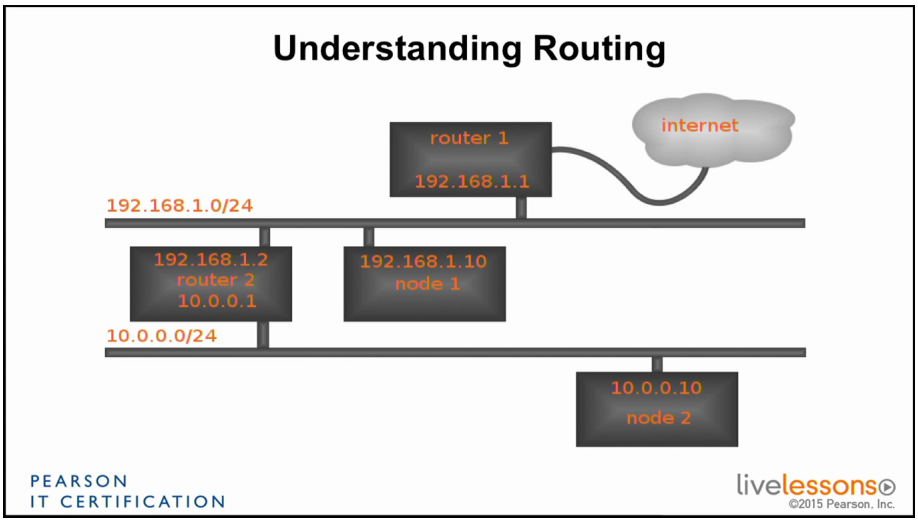

Figure 1.1: Sample Network

Here, we see two different networks - the `10.0.0.0/24` network connected to the inner `192.168.1.0/24` network via *router 2* (`10.0.0.1`), which in turn connects to the internet via the edge router with IP `192.168.1.1` - *router 1*.

For any packet headed to the internet on network 2, i.e., any packet originating from *node 2*, the default gateway will have to be *router 2* (`10.0.0.1`). This gets the packet on to the `192.168.1.0/24` network, where the default gateway is *router 1* (`192.168.1.1`), which passes it on to the internet.

However, when the packets originate from node 1 (`192.168.1.10`), there are two possible routes - if the packet is destined for the `10.0.0.0/24` network, then the gateway should be *router 2* (`192.168.1.2`). But if the packet is for any other network, then the default gate-

way of *router 1* (`192.168.1.1`) should be used. Thus, a static route should be defined on node 1 for the `10.0.0.0/24` network.

1.3 Setting up Static Routing

The most convenient way to set up static routes is to use `nmtui`. Let's assume we're setting up static routing for node 2 in our last example.

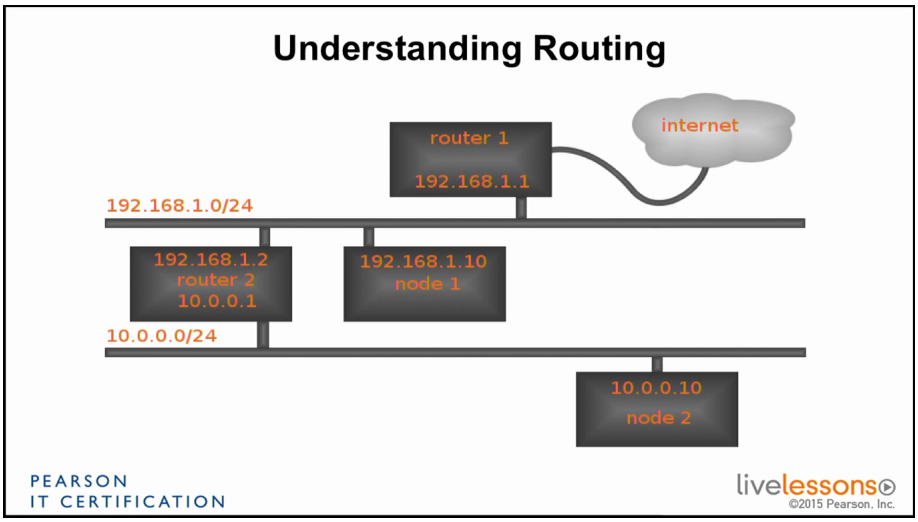

Figure 1.2: Network Diagram

We need to edit the existing connection to include the new static route. For this, we select the options: `Edit a Connection` → Select the connection to use → `Edit...` → `Routing` section → `Edit...` → `Add...` → Type the address of the network for which the static route will be defined in `Destination/Prefix` (with the Network ID and prefix, like, `10.0.0.0/24`) → Add the IP address of the router that leads to the network in the `Next Hop` section (`192.168.1.2` in our case).

The **metric** of the connection is how a router chooses which route to take when there are multiple routes available to another network. Thus, it's only useful when there are multiple routes available for the same network, and is irrelevant to us right now. We now choose `<Ok>` → `<Ok>` → `<Quit>`.

Note however, that the new route won't be added to the network configuration till either the connection is *refreshed* (by reactivating the connection) or the NetworkManager service is restarted. We could do this by `nmtui` → `Activate a Connection` → Select the connection which we edited → `Activate`. Now the output of `ip route show` will show the static route as well.

If the interface name was *ens33*, The `/etc/sysconfig/network-scripts` directory now has a new file called : `route-ens33` with the following contents:

```
1 ADDRESS0=10.0.0.0
2 NETMASK0=255.255.255.0
3 GATEWAY0=192.168.1.2
```

Note that the nmtui utility has translated the `/24` prefix from the **CIDR** (Classless Inter-Domain Routing) notation `10.0.0.0/24` to the standard Network IP and Network Masks, where `/24` translates to the network mask of `255.255.255.0`.

1.4 Understanding Network Bridges

A network bridge is a device that connects two or more networks to form one extended network. For example, an Ethernet bridge connects two or more LANs to create a unified, extended LAN. Virtual bridges are special purpose network interfaces used in virtualized environments.

Let us consider that the physical host has a NIC called `eno1`. The entire virtualized network in the diagram then has to communicate with any external networks via this interface. However, they can't all just send their packets to the driver of the NIC. Thus, they need a virtual bridge `virbr0`. There can be multiple virtual bridges too.

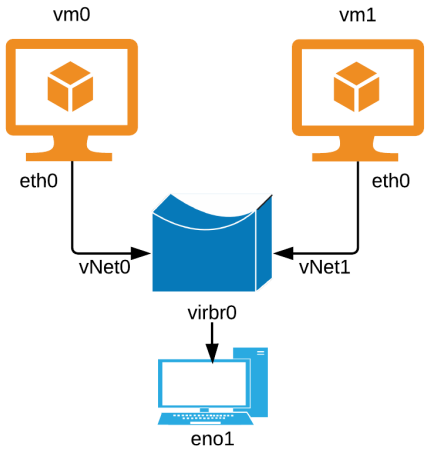

Figure 1.3: A virtualized network

The virtual bridge acts like a physical switch in the network and merely passes data between the networks. Note that it is incapable of routing decisions. All network traffic - even the traffic that originates from the physical KVM host are handled by it and thus, the virtual bridge decides who can send their packets at a specific moment.

Each of the virtual machines have their own virtualized Ethernet interface called `eth0` which have to be connected to an interface (port) on the virtual bridge. The virtual bridge names them `vnet0` and `vnet1` accordingly.

1.4.1 Working with Network Bridges

Let us consider a physical host with two KVM virtual machines running on it. Then, we can see their details using:

```
# virsh list --all
Id    Name                           State
----------------------------------------------------
3     vm0                            running
4     vm1                            running
```

Linux has an inbuilt layer 2 Ethernet bridge. This can be controlled using the `brctl` command. The status of the devices (VMs) connected to the bridge can be viewed with:

```
# brctl show
bridge name         bridge id               STP enabled       interfaces
virbr0              8000.525400683445       yes               virbr0-nic
                                                              vnet0
                                                              vnet1
```

The vnet0 and vnet1 interfaces are from the vm0 and vm1 virtual machines that are running on the host machine. The details of these interfaces can be seen with:

```
# ip link show
...
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 ↪  DEFAULT qlen 1000
link/ether 00:0c:29:d8:97:c2 brd ff:ff:ff:ff:ff:ff
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
 ↪  qlen 1000
link/ether 52:54:00:68:34:45 brd ff:ff:ff:ff:ff:ff
...
7: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master virbr0 state
 ↪  UNKNOWN mode DEFAULT qlen 1000
link/ether fe:54:00:0d:4a:d5 brd ff:ff:ff:ff:ff:ff
8: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master virbr0 state
 ↪  UNKNOWN mode DEFAULT qlen 1000
link/ether fe:54:00:21:93:04 brd ff:ff:ff:ff:ff:ff
```

The virtual bridge virbr0 connects several devices together: the virtual ethernet interfaces from the VMs, vmnet0 and vmnet1 to the external LAN via interface ens33, which is the NIC for the physical host. The virtual bridge only shows active interfaces connected to it, i.e., only when the VMs are running will they appear on the output of `brctl show`.

1.4.2 Difference between network device and interface

In terms of hardware, a device refers to the physical NIC that's connected to the host, while an interface refers to the physical port that an Ethernet cable is plugged into, i.e., the hardware Ethernet port. Back when each NIC had only one interface, the terms *device* and *hardware* meant the same thing. However, with the advent of NICs with multiple interfaces on the same NIC, for example dual or quad port configurations, interfaces refer to each separate hardware port that's capable of accepting a network cable. Thus, for a NIC with 4 hardware ports, the single device has 4 interfaces.

Linux however, doesn't see these interfaces as connected devices (unless configured to do so) and treat them like separate hardware devices, even though they're on the same card! So, there can be multiple interfaces per hardware device. However, from Linux's perspective, they're all separate network devices, thus making the terms *interface* and *device* synonymous.

1.5 Setting up Network Bridges

The roles of the interfaces on the bridge are defined by the connections (profiles) for the available interfaces. Thus, we generate new profiles for the interfaces that we want to act as slaves, and one connection for the interface that we intend to act as master.

1.5.1 Creating a slave interface on the bridge

The package needed to set up software bridges on RHEL 7 is called `bridge-utils`. To set up bridging all connected interfaces need to be disconnected and then connected to the bridge. The connected interface can be viewed with:

```
# nmcli dev show
GENERAL.DEVICE:                         ens33
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         00:0C:29:3B:B9:1C
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
GENERAL.CONNECTION:                     ens33
GENERAL.CON-PATH:
    ↪  /org/freedesktop/NetworkManager/ActiveConnection/1
WIRED-PROPERTIES.CARRIER:               on
IP4.ADDRESS[1]:                         10.0.99.11/24
IP4.GATEWAY:                            10.0.99.2
IP4.DNS[1]:                             10.0.99.2
IP6.ADDRESS[1]:                         fe80::f408:1ebf:7742:9fd8/64
...
```

Now, we disconnect the devices using:

```
# nmcli dev disconnect ens33
Device 'ens33' successfully disconnected.
```

Once the disconnection is complete, we can now start defining the bridge, by adding an interface connection with:

```
# nmcli con add con-name br0-port1 type bridge-slave master br0 ifname ens33
Warning: master='br0' doesn't refer to any existing profile.
Connection 'br0-port1' (3dee7b9b-6197-4eb7-be8d-46290361b9fd) successfully added.
```

In this command, we have created a new connection with the name `br0-port1` and connected it to the interface *ens33*, which refers to the hardware NIC of the host. We've set the connection type to **bridge-slave**, which refers to the fact that the details of the incoming connection to the *slave* interface (*ens33*) will be configured at the bridge. The master of the new slave connection is set to a connection called *br0* (which doesn't yet exist, leading to the warnings).

The advantage of the master-slave configuration in network bridges is that if there are many connected slave interfaces, we need not set up their properties individually, and the master (bridge) thus provides a central point of configuration. The above slave configuration has to be repeated for every slave interface that we wish to connect to the bridge.

1.5.2 Creating a master interface on the bridge

The connection for the master interface will determine the settings for all the slave interfaces connected to it. We create the master connection by setting the type to `bridge` (instead of `bridge-slave`, unlike the previous cases):

```
# nmcli con add con-name br0 type bridge ifname br0
Connection 'br0' (493c6288-7f40-4a9b-9ac0-aee7a7aebe71) successfully added.
# brctl show
bridge name        bridge id               STP enabled        interfaces
br0                8000.000000000000       yes
```

The NetworkManager must be restarted to be actually able to use any of this, since these only refer to the configuration in scripts `/etc/sysconfig/network-scripts` that need to be created by the NetworkManager.

```
# cd /etc/sysconfig/network-scripts/
# ls ifcfg-*
ifcfg-br0  ifcfg-br0-1  ifcfg-br0-port1  ifcfg-ens33  ifcfg-lo
```

The contents of the interface configuration file for the br0 master port:

```
DEVICE=br0
STP=yes
BRIDGING_OPTS=priority=32768
TYPE=Bridge
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=br0
UUID=5da1229d-27f1-4261-8491-e30046b9d03d
ONBOOT=yes
```

Since no information about IPs were provided, the boot protocol was chosen to be *DHCP* automatically. The slave interfaces only have the configuration:

```
TYPE=Ethernet
NAME=br0-port1
UUID=3dee7b9b-6197-4eb7-be8d-46290361b9fd
DEVICE=ens33
ONBOOT=yes
BRIDGE=br0
```

Since the configuration via the `nmcli` utility is hard to remember, it's man page has a link to the *nmcli-examples* man page, which has specific examples on setting up a bridge connection, as well as much more of the `nmcli` functionality.

1.6 Understanding Network Bonds and Teams

Both network bonds and teams accomplish roughly the same goal - the aggregation of links or network interfaces to form Link Aggregation Groups (LAG) or virtual links. This means several physical/logical interfaces can be combined to form a *team* that together fulfil a responsibility. Thus, one link may be set up as a primary connection to the WAN while the other may act as a backup or they both may be configured to act together while load balancing. Network bonding has been deprecated in RHEL 7 and thus we'll concentrate on Network teaming.

Network bonding used to perform the same responsibility as network teaming, but in the user space. Contrastingly, network teaming works with a kernel driver but also has a user space daemon, called **teamd**. This *teamd* daemon has several modes of operation called *runners*. These determine the function of the ports in the team and have possible values of: *broadcast, roundrobin, activebackup, loadbalance* and *lacp*.

Terms	Description
broadcast	Any packet is broadcast all over the interfaces.
roundrobin	The port which can transmit data is chosen in a roundrobin fashion.
activebackup	One of the interfaces stays active while the other is backup, ready to kick in the moment the active interface fails.
loadbalance	The network load (i.e., the packets in the network) is split between the interfaces so as to not overload any single interface.
lacp	Link Aggregation Control Protocol - allows formation of LAGs on a peer by automatically negotiating by transmitting LACP packets.

The command to control and manage teams is called `teamdctl`. Thus, to show the state of the team called *team0*, we'd use: `teamdctl team0 state`.

1.7 Configuring Network Teams

There are four parts to creating a team:

- Creating a team interface
- Determining the network configuration
- Assigning the port interfaces
- Bring team and port interfaces up and down respectively.

Once the above has been taken care of, the team connection can be verified with `teamdctl team0 status` (assuming *team0* is the name of the team).

1.7.1 Creating the team interface

We have to create the new team connection, preferably with the same interface name as the team:

```
1 # nmcli con add type team con-name team0 ifname team0 config '{"runner": {"name":
  ↪  "loadbalance"}}'
2 Connection 'team0' (d0eba200-591c-4ce3-a089-12d8a5692243) successfully added.
```

In this command, we have created a connection of type *team*, which indicates that it'll be a link aggregate. We provide a name to it called *team0* and connect it to an interface called team0 (which is logical - the interface that'll act as the aggregate of the member links). Finally, we provide the configuration as a JSON array: `'{"runner": {"name": "loadbalance"}}'`. This sets the team to act as a load balancer, and thus split the load of the packets over all the interfaces configured in the team.

1.7.2 Determining the network configuration

The team needs to be configured to use an IP address to use as its interface (i.e., team interface). This is specified using the CIDR (Classless Inter-Domain Routing) notation with a Network IP address and a prefix. From this both the Network ID and the Subnet mask can be determined.

```
# nmcli con mod team0 ipv4.addresses 10.0.0.10/24
# nmcli con mod team0 ipv4.method manual
```

Note that the `mod` command uses `ipv4` instead of `ip4`, unlike the `nmcli add` command. The IP assignment method also needs to be switched to manual since DHCP isn't involved here.

1.7.3 Assigning the port interfaces

Now that the master (team) interface has a port defined for it, we also need to assign the individual interfaces that are going to be slaves to the team. This is done using:

```
# nmcli con add type team-slave con-name team0-ens33 ifname ens33 master team0
Connection 'team0-ens33' (78e706bd-395c-456f-b16a-75430c48be2c) successfully added.
# nmcli con add type team-slave con-name team0-enss37 ifname ens37 master team0
Connection 'team0-enss37' (d17ad2f5-9a55-422a-ad5d-f4b327674393) successfully added.
```

The command defines two interfaces (*eth0* and *eth1*) to be slaves to the team interface. They are named according to the format `<teamName>-<interfaceName>`. Now we only need to define a master for them from which they can be controlled.

1.7.4 Bringing the team and port interfaces up/down

Once the above sections have been handled, the team is basically ready for operation. However, we still need to bring the physical devices (that are slaves to the team) to be disconnected and then reconnected as part of the team. Since we've already defined them as a part of the team, we just need to:

```
# nmcli con up team0
Connection successfully activated (master waiting for slaves) (D-Bus active path:
↪  /org/freedesktop/NetworkManager/ActiveConnection/7)
# nmcli dev disconnect ens33; nmcli dev dis ens37
Device 'ens33' successfully disconnected.
Device 'ens37' successfully disconnected.
```

The devices *eth0* and *eth1* needed to be disconnected because they're slaves to the team now, and thus, their operation should only be influenced by the team itself. Thus, there's no point in having them exist as separate individual devices (network interfaces).

1.7.5 Verifying the team connection

We can verify the team connection by:

```
# teamdctl team0 state
setup:
  runner: loadbalance
ports:
  ens33
    link watches:
      link summary: up
```

```
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  ens37
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
```

1.7.6 Creating a bridge based on Network Teams

When creating bridges based on network teams, it becomes important to switch off NetworkManager since they're incompatible. The team configuration file `ifcfg-team0` needs to be edited to add the line `BRIDGE=brteam0` has to be added to it to ask it to connect to the bridge device.

Since the team interfaces will be slaves to the connection provided by the team, it's important to ensure that there are no IP configurations in the `ifcfg-team0-port` files anymore.

Now we can manually create a configuration file for the bridge connection that has the contents:

```
DEVICE=brteam0
TYPE=Bridge
IPADDR0=192.168.122.100
PREFIX0=24
```

Now, the bridge on top of the team interface should be active and operational. Again, examples for these congigurations are present in the nmcli-examples man page, accessible with `man 5 nmcli-settings`.

1.8 Configuring IPv6

Since there is a shortage of unique IPv4 addresses when compared to the number of devices that are connected to the internet, IPv6 is the new standard for the IP addresses. Just like IPv4 addresses, it can be divided into two parts: the network ID and the host ID. However, in the case of the IPv6 addresses, the host ID contains the MAC address of the interface itself! A typical IPv6 address looks like:

```
fe80::2af4:9908:7092:34cb/64
```

In this, the network ID is `fe80` while the node part is the `2af4:9908:7092:34cb` with the prefix of `64` defining how long the network ID is. Thus, the computer listens for the Network ID, and then appends it's own MAC address to obtain a truly unique IPv6 address! For configuring connections with both IPv6 and IPv4 addresses, nmcli can be used as:

```
# nmcli con add con-name testCon type ethernet ip6 2001:db8:0:10::d000:310/64 gw6
↪  2001:db8:0:10::1 ip4 192.168.0.10/24 gw4 192.168.0.1
```

Again, for new connection just like for IPv4 connections where `nmcli con add` takes as argument the term **ip4** instead of *ipv4* (unlike the `con mod` command, which accepts *ipv4*), the `nmcli con add` needs an argument of **ip6** and not *ipv6*. Now to add a DNS server for it:

```
1 # nmcli con mod testCon +ipv6.dns 2001:4680:4680::8888
```