How to emulate a network using VirtualBox

July 4, 2016 - 27 Comments

VirtualBox is an open-source virtual machine manager and hypervisor that may also be used as a network emulator. In addition to creating and managing individual virtual machines, VirtualBox can connect virtual machines together to emulate a network of computers and network appliances such as routers or servers. VirtualBox works on the major computing platforms: Windows, MacOS, and Linux.

In this post, I offer a step-by-step tutorial showing how to use the VirtualBox graphical user interface to set up a network of six devices: three routers and three PCs. This tutorial will utilize some of the advanced functions supported by VirtualBox and provide you with the skills to set up a network of virtual machines on your own personal computer.

Required knowledge

I assume you, the reader, are already familiar with the VirtualBox GUI and have used it to create and run virtual machines on your personal computer, using default settings. I also assume you have a basic understanding of Linux shell commands, which will be needed to configure the Linux operating system running on the virtual routers and PCs.

If you need to refresh your knowledge about VirtualBox, the VirtualBox website provides a detailed user manual, and I have written a few posts featuring VirtualBox. See the list below:

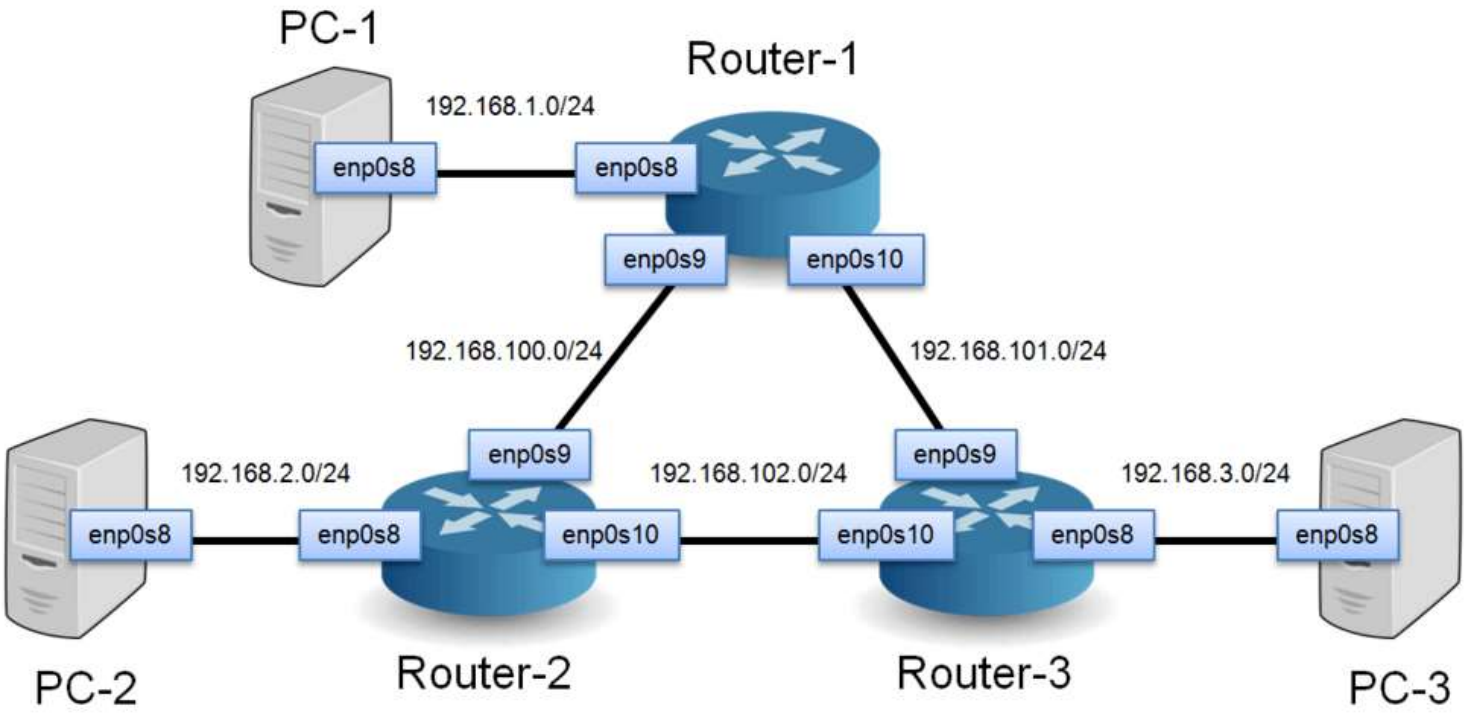
- Installing Debian Linux in a VirtualBox VM
- Installing dCore Linux in a VirtualBox VM
- Using VirtualBox Linked Clones in GNS3

Network topology

To build the emulated network, first create a network plan you can follow. VirtualBox does not have a drag-and-drop graphical user interface for creating networks of virtual machines so you must draw the network using another tool such as Microsoft PowerPoint, Visio, or open-source alternatives like LibreOffice Draw or Dia — or even pencil and paper.

Determine which nodes and ports connect to which networks before you start creating virtual machines. Plan how you will manage the emulated nodes. Once the network topology and IP network design is defined, build configuration plans (see the tables I use later in this post) and set up and debug the emulated network.

Create a small network of three routers, each of which is connected to a PC. The network topology you will create i- shown in the figure below:

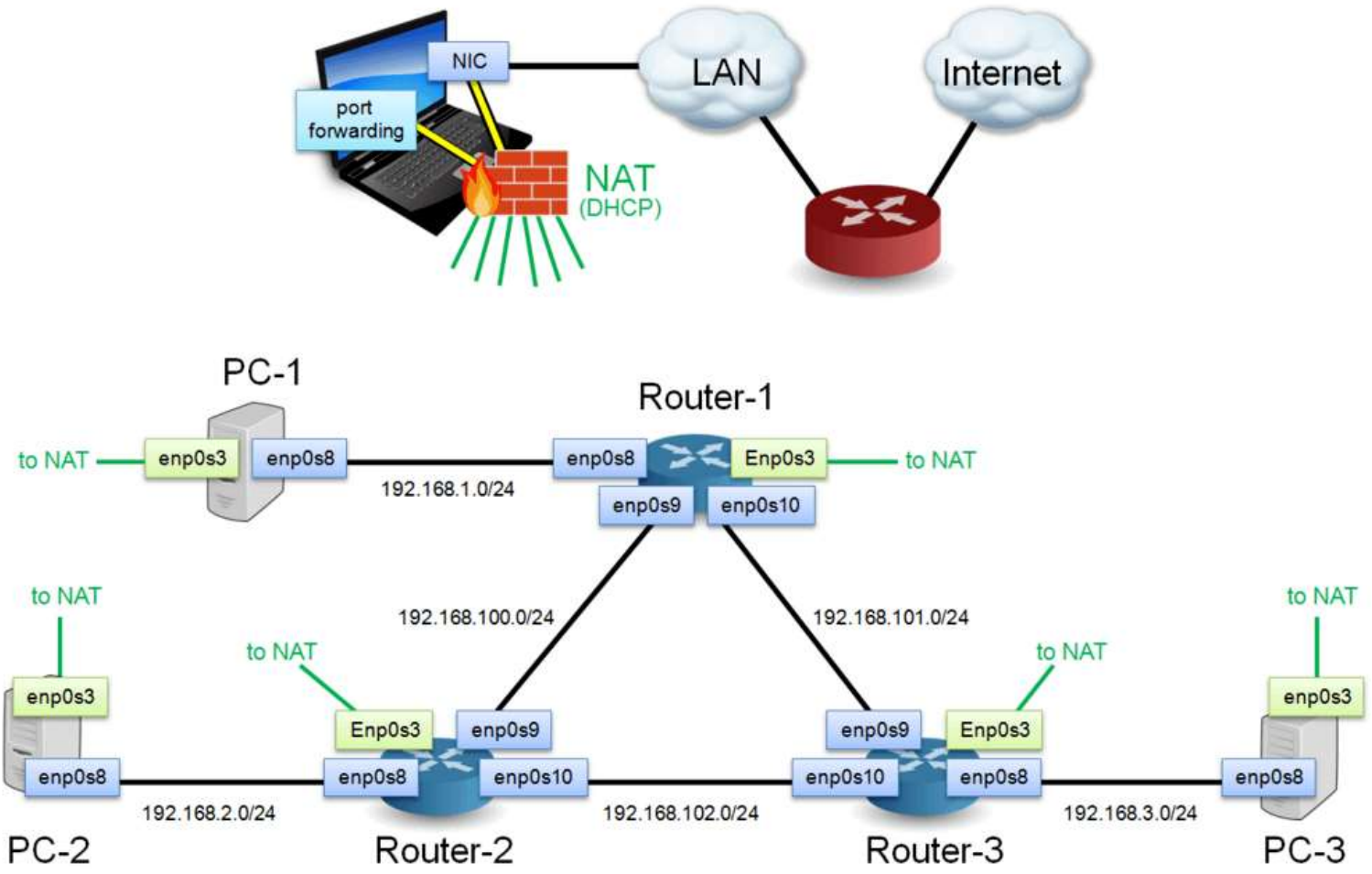


Test network topology

VirtualBox network topology

The VirtualBox network topology includes the interconnected guest virtual machines, the host computer, and external networks reachable from the host computer.

Each virtual machine is connected to other virtual machines by VirtualBox internal networks. I added a network adapter on each guest VM and attached it to the VirtualBox NAT interface to connect each guest VM to the host computer and to other external networks. I show the VirtualBox network topology in the figure below.



VirtualBox network with internal networks and a NAT management network

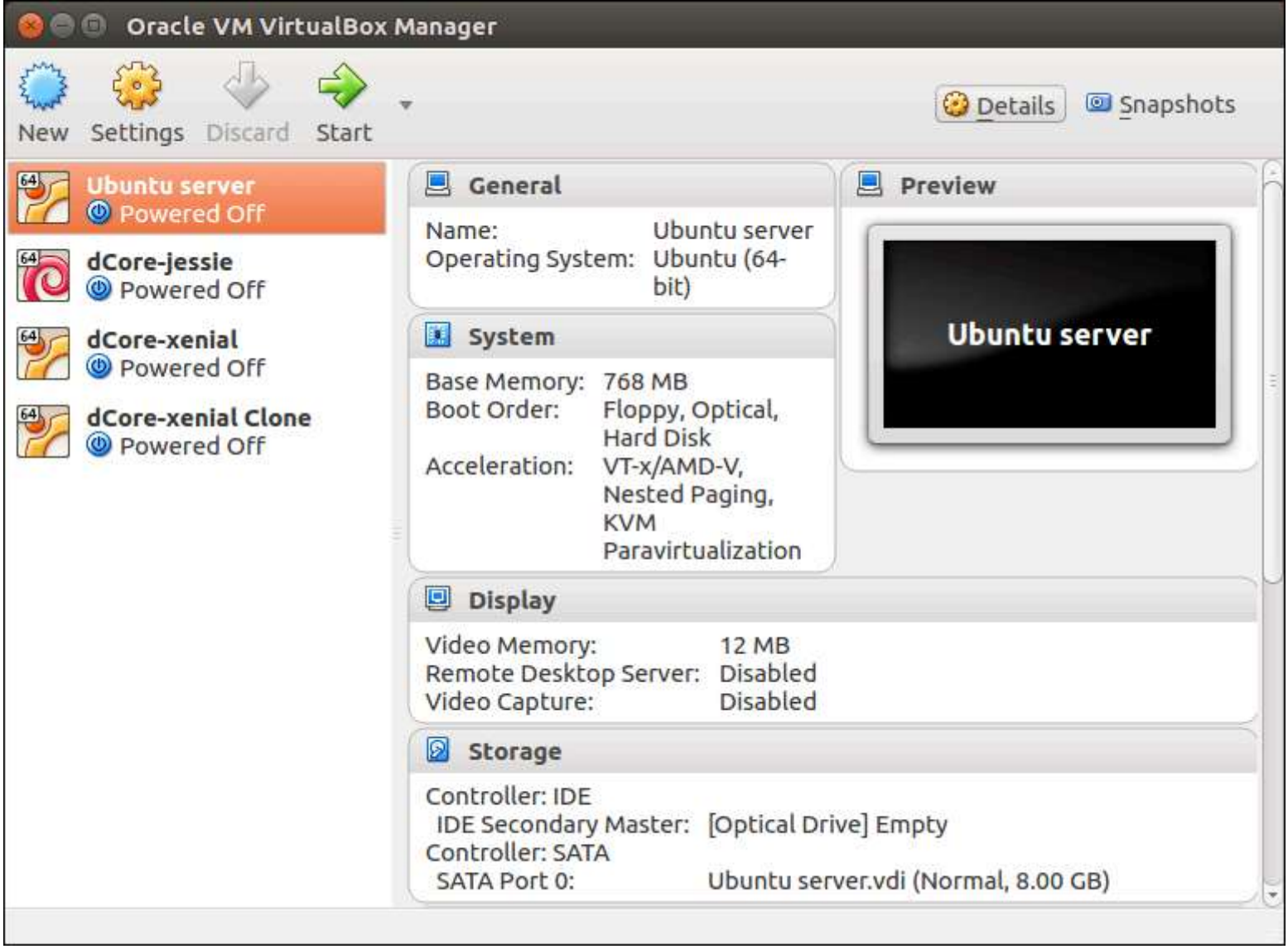
Everything in the above diagram — except for the LAN, the Internet and the router (colored red) that connects the LAN to the Internet — is running on your personal computer, represented by the laptop computer in the network diagram. Your personal computer is connected to a local area network and to the Internet via a router.

Create base virtual machines

To create the network topology, you must first create a new VM. In this case, I have already created a VM in VirtualBox. If you need to know how to install a VM in VirtualBox, please see my post about installing a Linux system in a VirtualBox VM.

The base VM

I installed Ubuntu Server 16.04 in this example. I used all the default configurations. See below that a virtual machine named *Ubuntu server* appears in the VirtualBox Manager window.



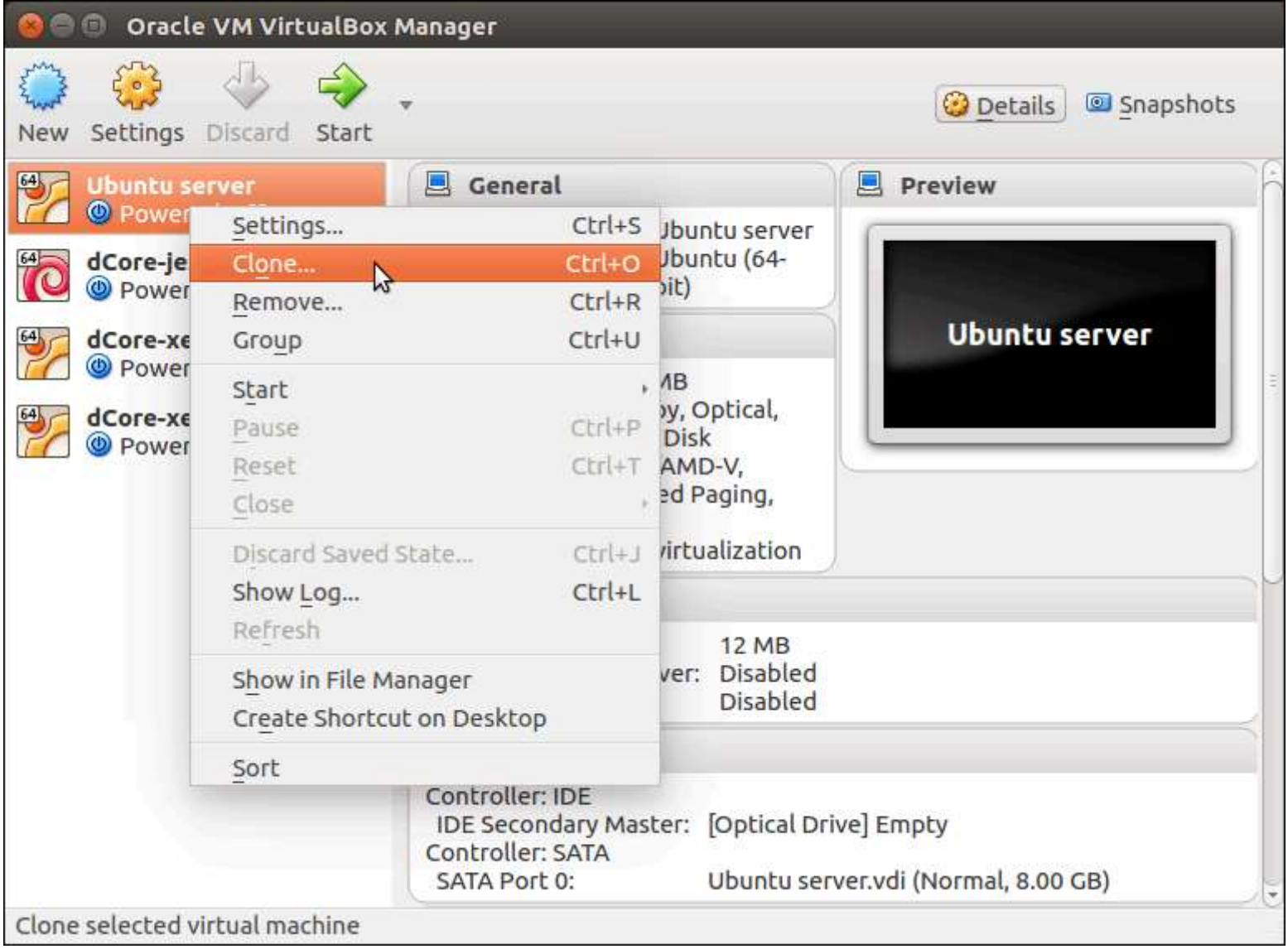
Virtual Machine *Ubuntu server* ready to use

Note: I used the default hostname for the server. The hostname is *ubuntu*. If you chose a different hostname, you will need to modify some of the commands I list later in this tutorial.

Clone virtual machines

Cloning a virtual machine is the easiest way to create more guest virtual machines on the host computer.

To create the PC and Routers for our network emulation, clone the *Ubuntu server* VM you created in the previous step. Right-click on *Ubuntu server* and select *Clone* from the menu.



Clone the virtual machine

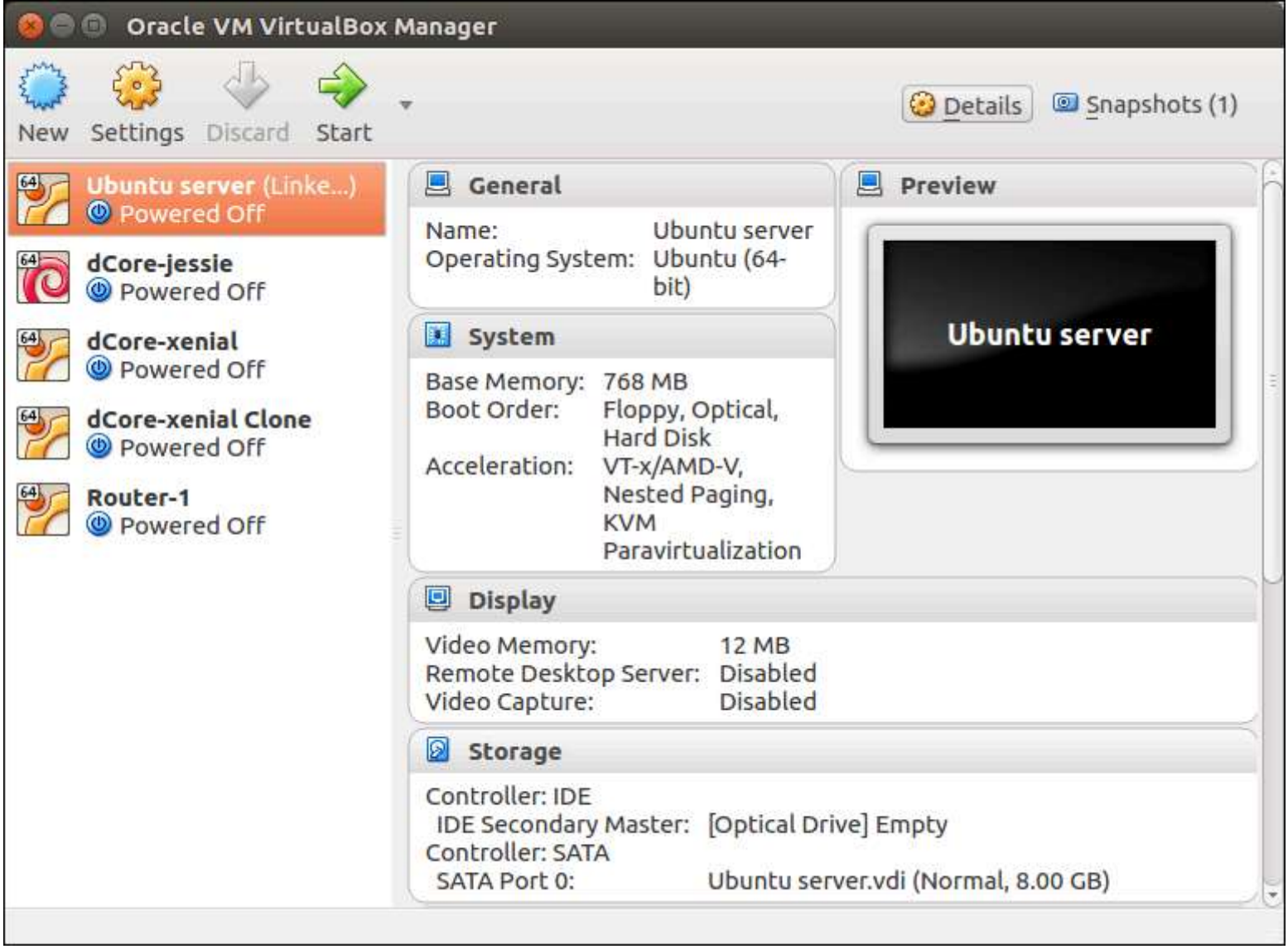
In the dialogue box, enter the new VM name and be sure to select the check box to *Reinitialize the MAC address of all network cards*. You must ensure that the MAC addresses will be different on each cloned VM.

Name the VM and reinitialize the MAC addresses

Choose *Linked Clone* in the next dialogue box. This will keep the cloned VM file size small.

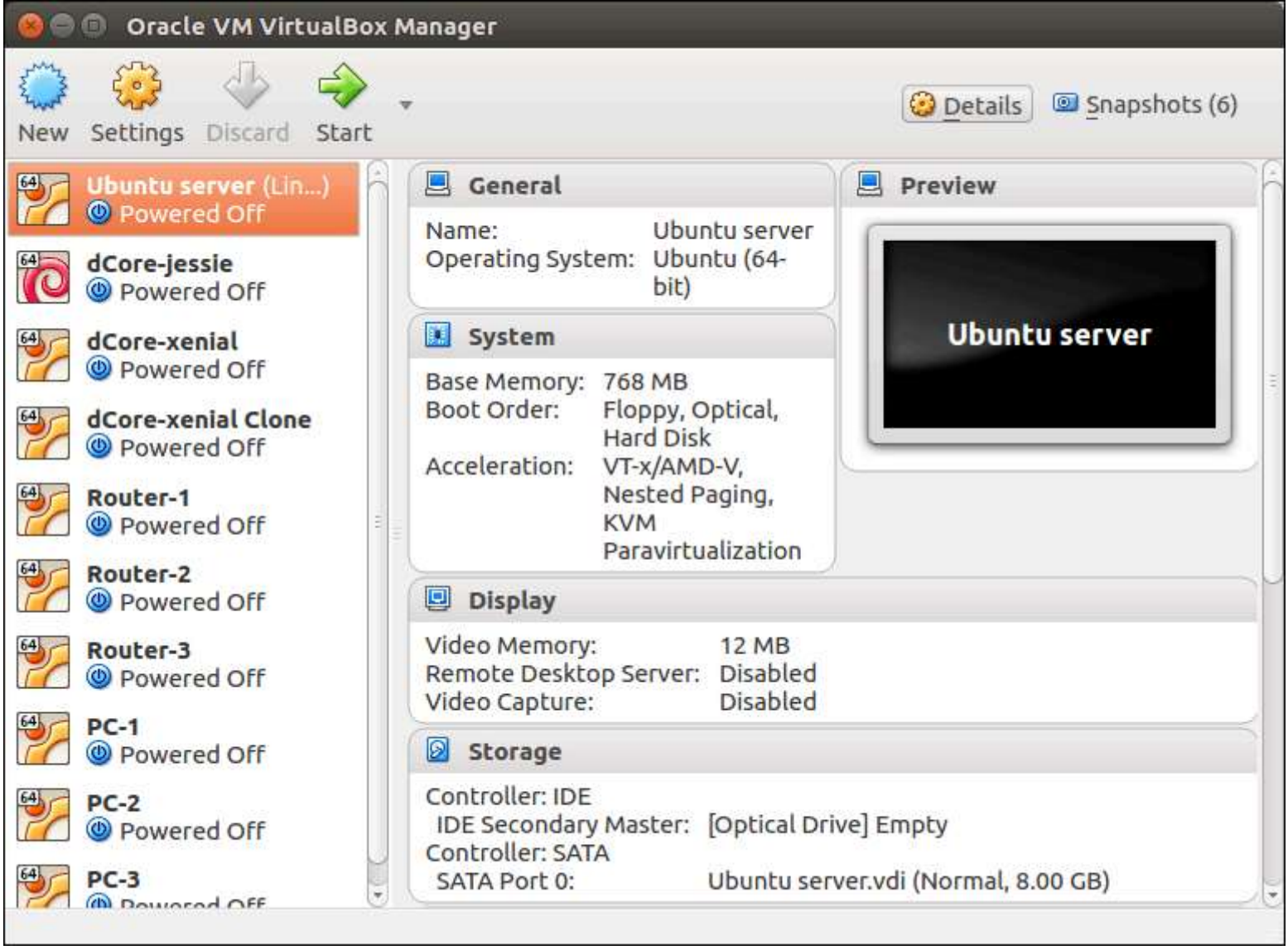
Select linked clone

Click the *Clone* button and the new VM will appear in the VirtualBox Manager window.



First linked clone created

Repeat the steps for each VM you require. In this case, create a total of six virtual machines, each of which is a linked clone of the *Ubuntu server* VM. Choose virtual machine names to match the node names defined in the network diagram.



Six linked-clone VMs created

Now you have six virtual machines. Each VM needs to be set up with network interfaces and connected to VirtualBox internal networks to create a network topology.

Create VirtualBox internal networks

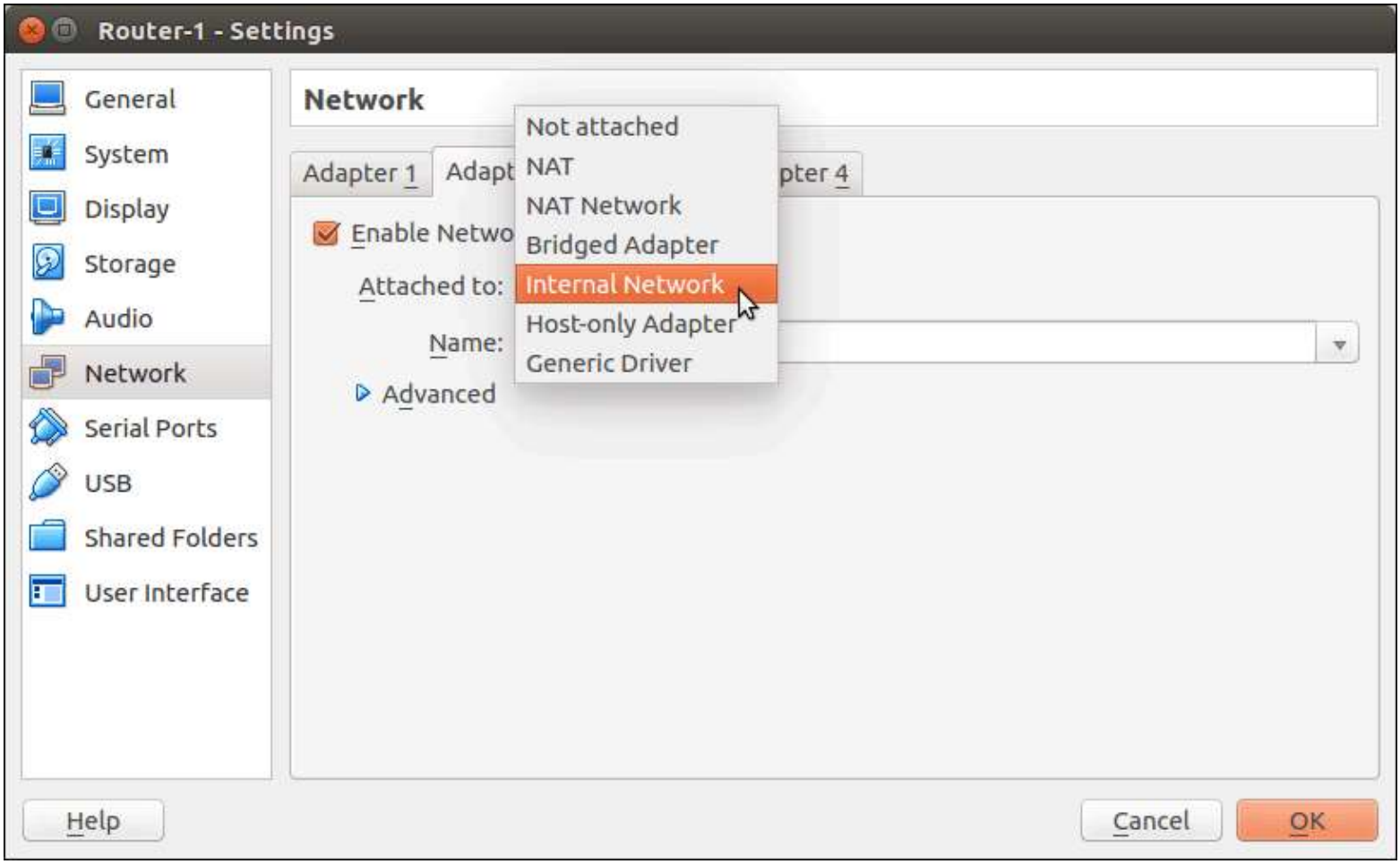
The VirtualBox graphical user interface supports only four network adapters for each VM. This limits the complexity of network scenarios you can create. Fortunately, VirtualBox really supports up to thirty-six network adapters per VM. These additional network adapters may be configured using the VirtualBox command-line interface, which is a topic for another post. For now, limit yourself to using the four adapters supported on each VM by the VirtualBox GUI.

Each network adapter may be enabled or disabled. If enabled, the adapter may be configured to connect to one of the many different types of interfaces provided by VirtualBox.

To connect two virtual machines to each other, use the *Internal Network* interface type.

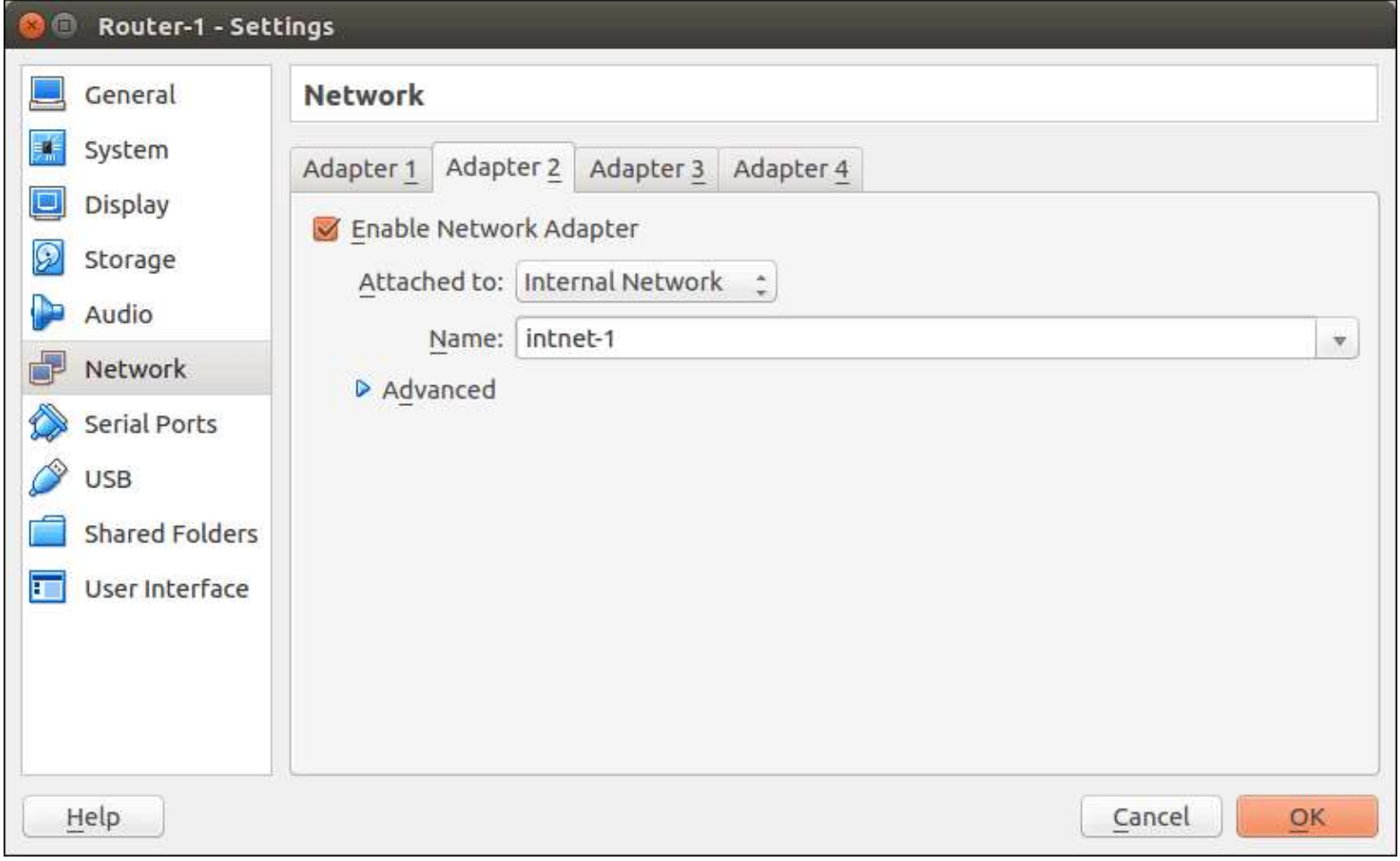
Select one of the virtual machines in the VirtualBox Manager window and click on *Settings*. Then, in the settings wi , click on *Network*. In the example below, you will configure Network Adapter 2 on the *Router-1* virtual machine.

Click on the *Enable Network Adapter* check box, if it is not already checked. Then click on *Attached To* and select *internal Network*.



Select internal network

Next, give the internal network a name. The name must match with the name configured on the corresponding network adapter on other the VM to be connected to this VM.



Enter network name

Repeat this process for each node. The routers each use three of the four available network adapters to connect to internal networks. The PCs each use one network adapter to connect to internal networks.

I used the internal network names shown in the table below to create point-to-point connections between each VM in the network topology.

Node	VirtualBox interface	VirtualBox Network Type	Name
PC-1	Adapter 2	Internal	intnet-1
PC-2	Adapter 2	Internal	intnet-2
PC-3	Adapter 2	Internal	intnet-3
Router-1	Adapter 2	Internal	intnet-1
	Adapter 3	Internal	intnet-100
	Adapter 4	Internal	intnet-101
Router-2	Adapter 2	Internal	intnet-2

	Adapter 3	Internal	intnet-100
	Adapter 4	Internal	intnet-102
Router-3	Adapter 2	Internal	intnet-3
	Adapter 3	Internal	intnet-101
	Adapter 4	Internal	intnet-102

Create management network

By default, VirtualBox connects the first network adapter on each virtual machine to the VirtualBox NAT interface. I use the VirtualBox NAT interface as a "management network" that enables each guest node to connect to external networks and, with port forwarding enabled, to the host computer's operating system.

TCP port forwarding

The VirtualBox NAT interface is a NAT firewall that connects guest virtual machines to the host computer's local area network. It supports DHCP configuration of IP addresses.

Because the virtual machines are hidden behind a NAT firewall, the host computer cannot initiate connections to them. To connect from the host computer to the virtual machines using SSH, you must set up TCP port forwarding on each virtual machine.

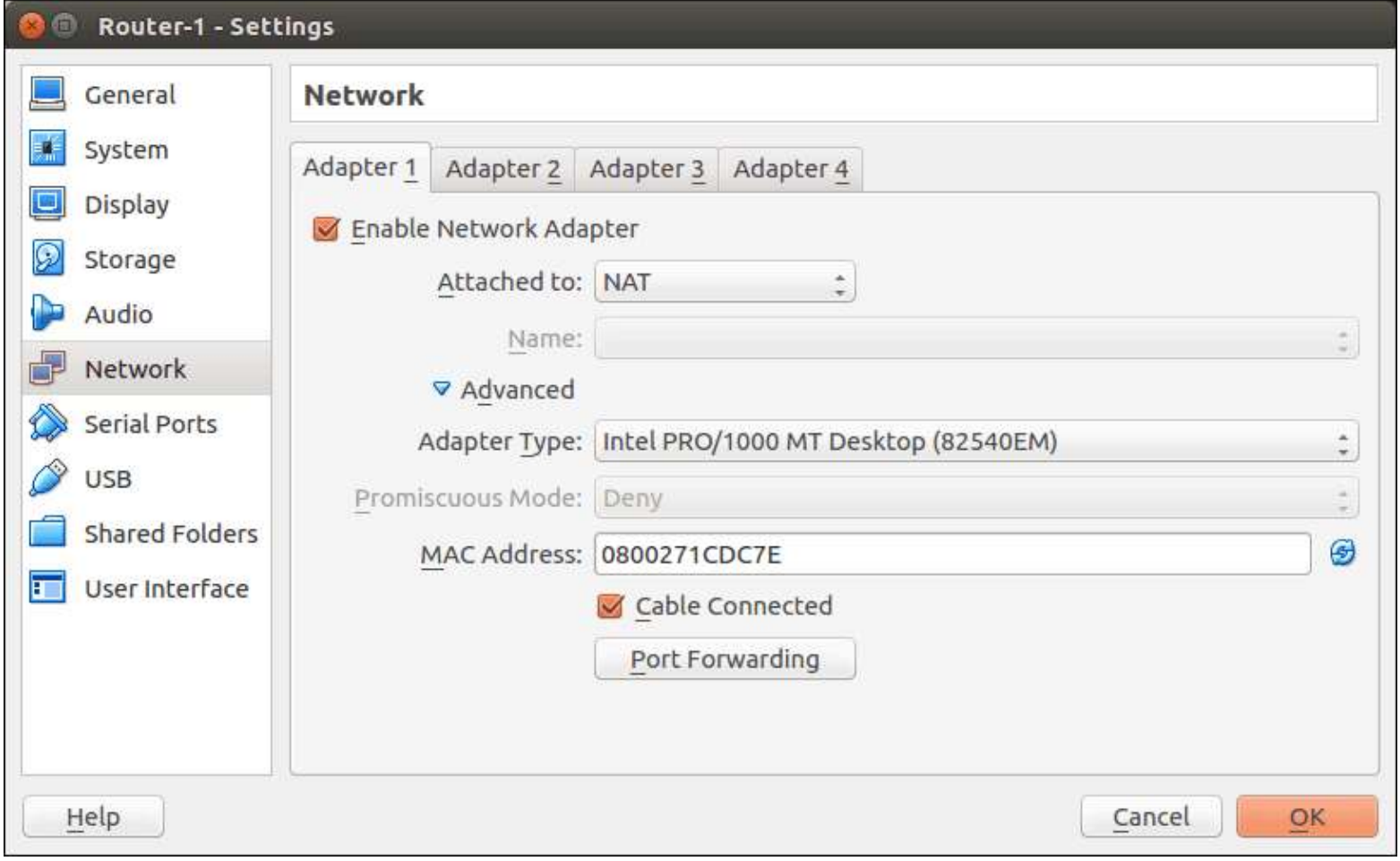
TCP port forwarding creates a hole in the NAT firewall through which the host computer or other clients from the local area network may initiate connections to the virtual machines.

The default SSH port on each guest virtual machine is TCP port 22. Map unused TCP port numbers on the host computer to port 22 on each guest virtual machine. Any unassigned or unreserved TCP port numbers may be used on the host computer. I prefer to use TCP port numbers between between 14415 and 14935 which provides 520 contiguous unassigned TCP port numbers ((Reference: http://stackoverflow.com/questions/10476987/best-tcp-port-number-range-for-internal-applications)).

To see all assigned TCP port numbers, see http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

Configure port forwarding on NAT interfaces

On each virtual machine, click on *Settings*, then click on the *Network* tab in the settings window. Select the tab for *Adapter 1*. Expand the *Advanced* network panel and click on *Port Fowarding*.

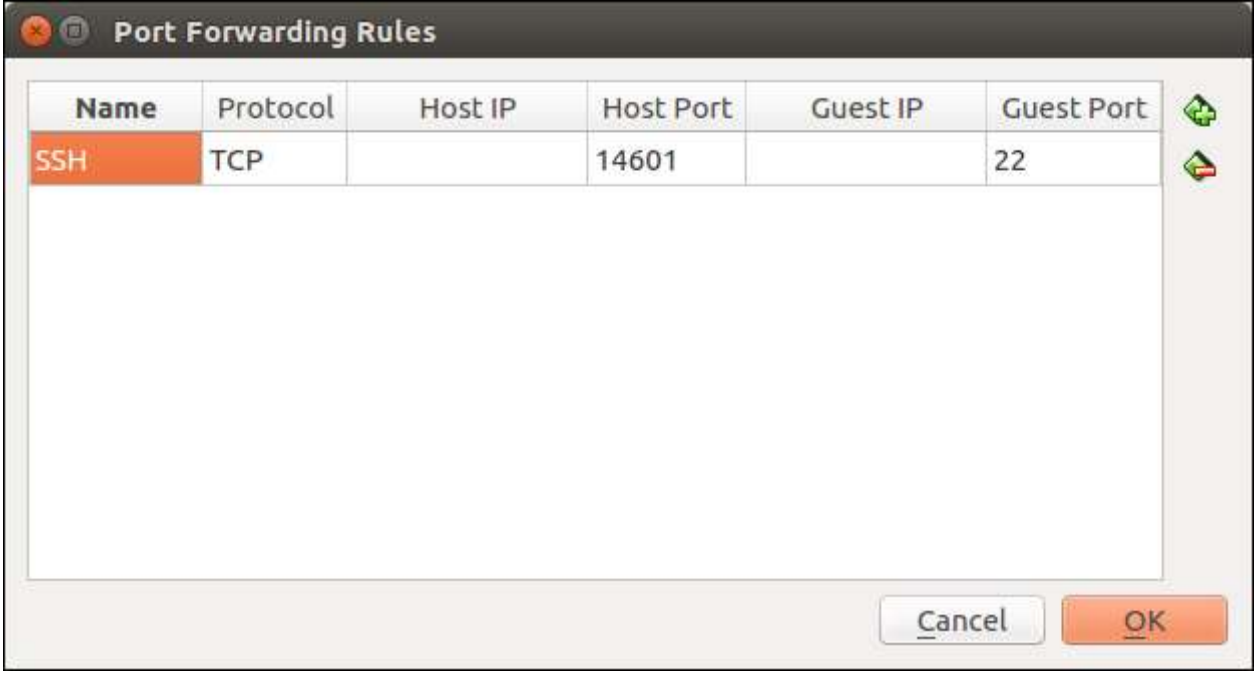


Advanced settings: Click on Port Forwarding

The Port Forwarding Rules window appears. Click on the green plus sign to add a new rule.

Port forwarding window

Give the rule a name. Any name may be used. I call the rule "SSH". The protocol is "TCP". Leave the IP address fields blank. The Guest Port is "22". The Host Port is any TCP port number available on the host computer. In this example use port number "14601".



Router-1 reachable using port 14601

Repeat the process and set up NAT interfaces with port forwarding on each virtual machine. To make it easier to remember port numbers, I assigned TCP port numbers to PCs starting with port number 14501 and I assigned port numbers to routers starting with port number 14601.

The table below shows the NAT interfaces on each machine and the TCP port forwarding rules for each interface.

Network Node	Interface	Rule Name	Host IP	Host Port	Guest IP	Guest Port
PC-1	Adapter 1	SSH	(blank)	14501	(blank)	22
PC-2	Adapter 1	SSH	(blank)	14502	(blank)	22
PC-3	Adapter 1	SSH	(blank)	14503	(blank)	22
Router-1	Adapter 1	SSH	(blank)	14601	(blank)	22
Router-2	Adapter 1	SSH	(blank)	14602	(blank)	22
Router-3	Adapter 1	SSH	(blank)	14603	(blank)	22

When I need to connect to a virtual network node from my host computer, I use IP address of the host computer's loopback interface (or just hostname *localhost*) and the Host Port number in the table listed above. You cannot use VM's IP addresses because it is hidden behind the NAT Firewall. Also, the DHCP server built into VirtualBox's NAT interface will assign the same IP address to each VM's attached network adapter. VirtualBox isolates each management interface so this is not a problem and the NAT function ensures that each VM appears to have a different IP on the LAN side of the NAT.

Configuring management interface on virtual machines

All the virtual machines used in this tutorial are clones of a base virtual machine. I created the base virtual machine, *Ubuntu Server*, while its first network adapter was connected to the NAT interface, which is the default configuration for VMs in VirtualBox. The Ubuntu Server installation scripts configured the system to use DHCP on the first ethernet interface *enp0s3*. So the base virtual machine gets IP configuration from the VirtualBox NAT interface's built-in DHCP server.

Each clone VM you created inherits the same configuration from the base VM so each VM should already be have interface *enp0s3* set up and running. You do not need to modify any configuration files on the virtual machines to enable them to connect to the NAT interface ((If you want to use a different network adapter for the NAT interface, edit the */etc/network/interfaces* file and restart the networking service or reboot the VM)).

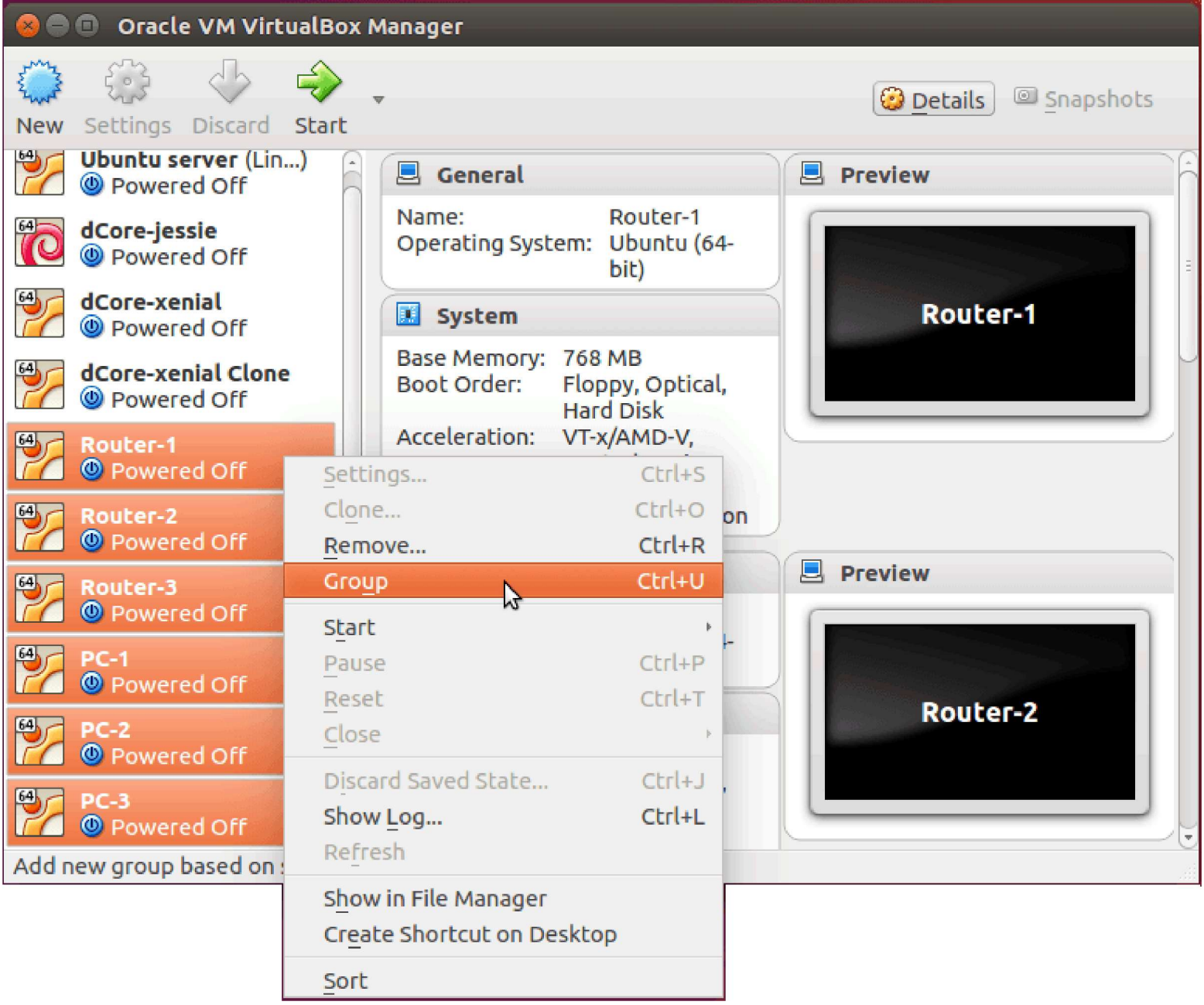
Start virtual machines

Now you may start the network emulation scenario by starting all the virtual machines.

As you can see in the figures below, you have a lot of virtual machines in the VirtualBox VM Manager. You need a way to keep track of the virtual machines created for the network emulation project so you don't lose track.

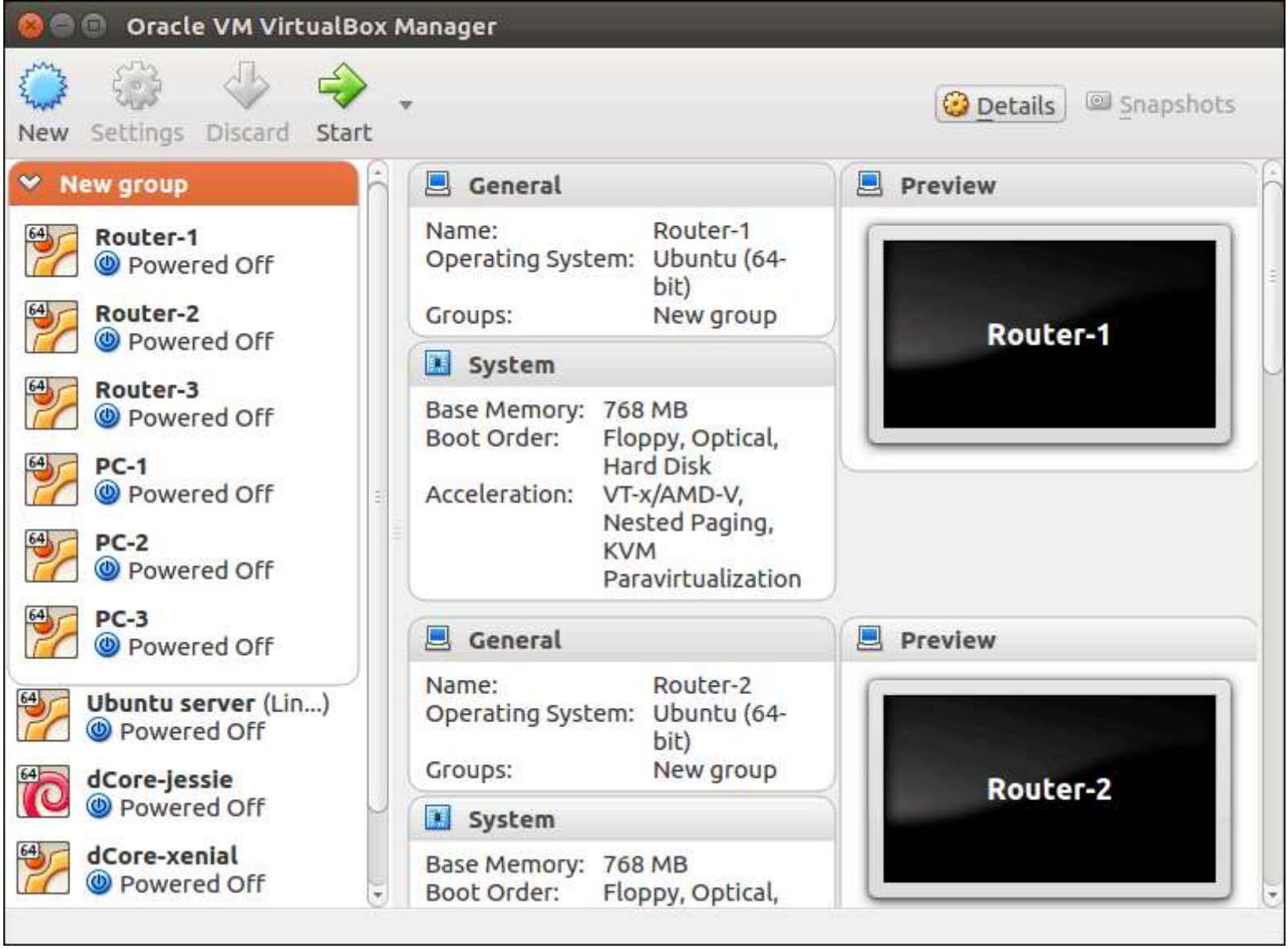
VirtualBox allows you to group virtual machines together. Set up the virtual machines created for the network emulation scenario in a group so you can start them all together and so you do not mix them up with other VMs you may have defined in the VirtualBox GUI.

To group VMs in VirtualBox, hold down the *Shift* key and select each VM that will be included in the group. Then right-click on the selected VMs and select *Group* from the menu.



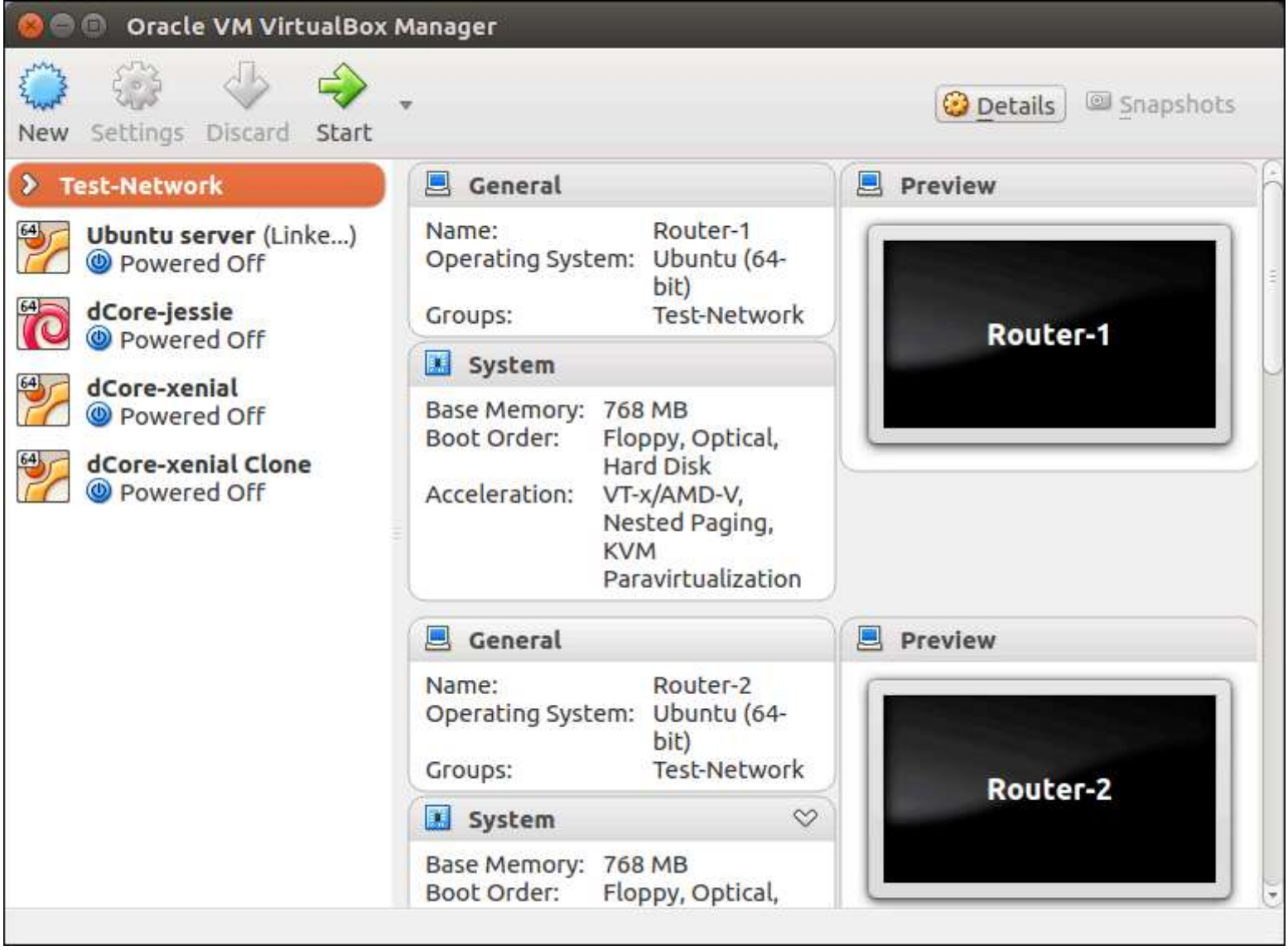
Group VMs together

VirtualBox draws a box around the group and gives the group a name, “New group”.



Group of VMs

Change the name of the group by double-clicking on the group name and typing a new name. You may also collapse the group to hide its contents by clicking on the small chevron icon to the left of the group name.



Group collapsed

When you select the group by clicking on the group name, you may apply VirtualBox commands like *Start* or *Stop* to the entire group. This makes it easy to start your network emulation scenario quickly.

In this example, select the group and then click on the green arrow to start the network emulation.

Connect to each virtual machine

Use SSH to log into virtual machines after you start them in the network emulation scenario. It will take a few minutes for all the virtual machines to start.

To log into any running virtual machine, use the host computer's IP address and the host port number assigned to the virtual machine:

```
$ ssh -l <userid> -p <port number> <IP address>
```

The `-l` option specifies the userid used to login to the node. In this case, when I installed the guest operating system on each node, I chose the userid *brian* so that is the userid I used in the SSH command.

The `-p` option specified the host port number. The host port is a TCP port currently listening on the host computer that will forward traffic to port 22 on the associated virtual machine. I use the host port numbers I assigned for the virtual machines in the table above.

I use `localhost` as the IP address because I am running the command on the host computer. Alternatively, I could use the host computer's loopback address *127.0.0.1*.

Open six terminal windows, one for each VM. In each window, use SSH to connect to a different VM. I enter the commands shown below into each VM's terminal window (or use Putty if you run Microsoft Windows):

Terminal	Virtual Machine	Command
1	PC-1	ssh -l brian -p 14501 localhost
2	PC-2	ssh -l brian -p 14502 localhost
3	PC-3	ssh -l brian -p 14503 localhost
4	Router-1	ssh -l brian -p 14601 localhost
5	Router-2	ssh -l brian -p 14602 localhost
6	Router-3	ssh -l brian -p 14603 localhost

The first Ethernet interface on each virtual machine is already configured to connect to a DHCP server so you should be able to SSH into each VM using the commands in the table above. If SSH will not work, check the IP configuration of the first Ethernet interface, *enp0s3*, on each virtual machine.

Configure and test network nodes

Now that all the virtual machines are running, configure their network interfaces and routing protocols.

Network configuration

Each terminal is now connected to a Linux shell on each virtual machine. Configure the network interfaces on each machine. On the routers, you also need to install routing software and enable networking protocols.

Using the network topology as a guide, make a table of IP addresses to be used to configure the ports on each virtual machine. In this example, I used the table shown below:

Node	Linux interface name	IP address to be assigned
PC-1	enp0s8	192.168.1.1/24
	enp0s3	DHCP
PC-2	enp0s8	192.168.2.1/24
	enp0s3	DHCP
PC-3	enp0s8	192.168.3.1/24
	enp0s3	DHCP
Router-1	enp0s8	192.168.1.254/24
	enp0s9	192.168.100.1/24
	enp0s10	192.168.101.2/24
	enp0s3	DHCP
Router-2	enp0s8	192.168.2.254/24
	enp0s9	192.168.100.2/24
	enp0s10	192.168.102.2/24
	enp0s3	DHCP
Router-3	enp0s8	192.168.3.254/24
	enp0s9	192.168.101.1/24
	enp0s10	192.168.102.1/24
	enp0s3	DHCP

See below for the configuration commands you may copy-and-paste into each VM's terminal window to set up the network.

See my post about how to build a network of Linux routers using quagga if you need explanations about how these commands work.

PC-1

On PC-1, change the hostname, add the interface configuration to the network interfaces file and set up a static route:

```
sudo su
```

Enter your password. Then copy and paste the following commands into the terminal windows. We use the here documents feature of the bash shell to redirect the pasted block of input to the bash shell command line:

```
 bash <<EOF2
sed -i 's/ubuntu/pc1/g' /etc/hostname
sed -i 's/ubuntu/pc1/g' /etc/hosts
hostname pc1
cat >> /etc/network/interfaces << EOF
auto enp0s8
iface enp0s8 inet static
   address 192.168.1.1
   netmask 255.255.255.0
up route add -net 192.168.0.0/16 gw 192.168.1.254 dev enp0s8
EOF
/etc/init.d/networking restart
exit
EOF2
```

Reboot the node:

```
sudo reboot
```

Then log back into the node from the host computer using SSH, using the SSH command shown above.

```
ssh -l brian -p 14501 localhost
```

PC-2

On PC-2, change the hostname, add the interface configuration to the network interfaces file and set up a static route. Copy-and-paste the following commands into the PC-2 terminal window:

```
sudo su
```

Enter your password. Then copy and paste the following commands into the terminal window:

```
 bash <<EOF2
sed -i 's/ubuntu/pc2/g' /etc/hostname
sed -i 's/ubuntu/pc2/g' /etc/hosts
hostname pc2
cat >> /etc/network/interfaces << EOF
auto enp0s8
iface enp0s8 inet static
   address 192.168.2.1
   netmask 255.255.255.0
up route add -net 192.168.0.0/16 gw 192.168.2.254 dev enp0s8
EOF
/etc/init.d/networking restart
exit
EOF2
```

Reboot the node:

```
sudo reboot
```

Then log back into the node from the host computer using SSH, using the SSH command shown above.

```
ssh -l brian -p 14502 localhost
```

PC-3

On PC-3, change the hostname, add the interface configuration to the network interfaces file and set up a static route. Copy-and-paste the following commands into the PC-1 terminal window:

```
sudo su
```

Enter your password. Then copy and paste the following commands into the terminal window:

```
 bash <<EOF2
sed -i 's/ubuntu/pc3/g' /etc/hostname
sed -i 's/ubuntu/pc3/g' /etc/hosts
hostname pc3
cat >> /etc/network/interfaces << EOF
auto enp0s8
iface enp0s8 inet static
   address 192.168.3.1
   netmask 255.255.255.0
up route add -net 192.168.0.0/16 gw 192.168.3.254 dev enp0s8
EOF
/etc/init.d/networking restart
exit
EOF2
```

Reboot the node:

```
sudo reboot
```

Then log back into the node from the host computer using SSH, using the SSH command shown above.

```
ssh -l brian -p 14503 localhost
```

Router-1 copy-and-paste shell commands

On Router-1, change the hostname, install quagga, and configure OSPF on the router's interfaces. Copy-and-paste the following commands into the Router-1 terminal window:

```
sudo su
```

Enter your password. Then copy and paste the following commands into the terminal window:

```
 bash <<EOF2
sed -i 's/ubuntu/router1/g' /etc/hostname
sed -i 's/ubuntu/router1/g' /etc/hosts
hostname router1
```

```
apt-get update
apt-get install quagga quagga-doc traceroute
cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf
chown quagga.quaggavty /etc/quagga/*.conf
chmod 640 /etc/quagga/*.conf
sed -i s'/zebra=no/zebra=yes/' /etc/quagga/daemons
sed -i s'/ospfd=no/ospfd=yes/' /etc/quagga/daemons
echo 'VTYSH_PAGER=more' >>/etc/environment
echo 'export VTYSH_PAGER=more' >>/etc/bash.bashrc
cat >> /etc/quagga/ospfd.conf << EOF
interface enp0s8
interface enp0s9
interface enp0s10
interface lo
router ospf
 passive-interface enp0s8
 network 192.168.1.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
line vty
EOF
cat >> /etc/quagga/zebra.conf << EOF
interface enp0s8
 ip address 192.168.1.254/24
 ipv6 nd suppress-ra
interface enp0s9
 ip address 192.168.100.1/24
 ipv6 nd suppress-ra
interface enp0s10
 ip address 192.168.101.2/24
 ipv6 nd suppress-ra
interface lo
ip forwarding
line vty
EOF
/etc/init.d/quagga start
exit
EOF2
```

Reboot the node:

```
sudo reboot
```

Then log back into the node from the host computer using SSH, using the SSH command shown above.

```
ssh -l brian -p 14601 localhost
```

Router-2

On Router-2, change the hostname, install quagga, and configure OSPF on the router's interfaces. Copy-and-paste the following commands into the Router-2 terminal window:

```
sudo su
```

Enter your password. Then copy and paste the following commands into the terminal window:

```
 bash <<EOF2
sed -i 's/ubuntu/router2/g' /etc/hostname
sed -i 's/ubuntu/router2/g' /etc/hosts
hostname router2
apt-get update
apt-get install quagga quagga-doc traceroute
cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf
chown quagga.quaggavty /etc/quagga/*.conf
chmod 640 /etc/quagga/*.conf
sed -i s'/zebra=no/zebra=yes/' /etc/quagga/daemons
sed -i s'/ospfd=no/ospfd=yes/' /etc/quagga/daemons
echo 'VTYSH_PAGER=more' >>/etc/environment
echo 'export VTYSH_PAGER=more' >>/etc/bash.bashrc
cat >> /etc/quagga/ospfd.conf << EOF
interface enp0s8
interface enp0s9
interface enp0s10
interface lo
router ospf
 passive-interface enp0s8
 network 192.168.2.0/24 area 0.0.0.0
 network 192.168.100.0/24 area 0.0.0.0
```

```
 network 192.168.102.0/24 area 0.0.0.0
line vty
EOF
cat > /etc/quagga/zebra.conf << EOF
interface enp0s8
 ip address 192.168.2.254/24
 ipv6 nd suppress-ra
interface enp0s9
 ip address 192.168.100.2/24
 ipv6 nd suppress-ra
interface enp0s10
 ip address 192.168.102.2/24
 ipv6 nd suppress-ra
interface lo
ip forwarding
line vty
EOF
/etc/init.d/quagga start
exit
EOF2
```

Reboot the node:

```
sudo reboot
```

Then log back into the node from the host computer using SSH, using the SSH command shown above.

```
ssh -l brian -p 14602 localhost
```

Router-3

On Router-3, change the hostname, install quagga, and configure OSPF on the router's interfaces. Copy-and-paste the following commands into the Router-3 terminal window:

```
sudo su
```

Enter your password. Then copy and paste the following commands into the terminal window:

```
 bash <<EOF2
sed -i 's/ubuntu/router3/g' /etc/hostname
sed -i 's/ubuntu/router3/g' /etc/hosts
hostname router3
apt-get update
apt-get install quagga quagga-doc traceroute
cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf
chown quagga.quaggavty /etc/quagga/*.conf
chmod 640 /etc/quagga/*.conf
sed -i s'/zebra=no/zebra=yes/' /etc/quagga/daemons
sed -i s'/ospfd=no/ospfd=yes/' /etc/quagga/daemons
echo 'VTYSH_PAGER=more' >>/etc/environment
echo 'export VTYSH_PAGER=more' >>/etc/bash.bashrc
cat >> /etc/quagga/ospfd.conf << EOF
interface enp0s8
interface enp0s9
interface enp0s10
interface lo
router ospf
 passive-interface enp0s8
 network 192.168.3.0/24 area 0.0.0.0
 network 192.168.101.0/24 area 0.0.0.0
 network 192.168.102.0/24 area 0.0.0.0
line vty
EOF
cat > /etc/quagga/zebra.conf << EOF
interface enp0s8
 ip address 192.168.3.254/24
 ipv6 nd suppress-ra
interface enp0s9
 ip address 192.168.101.1/24
 ipv6 nd suppress-ra
interface enp0s10
 ip address 192.168.102.1/24
 ipv6 nd suppress-ra
interface lo
ip forwarding
line vty
EOF
/etc/init.d/quagga start
```

```
exit
EOF2
```

Reboot the node:

```
sudo reboot
```

Then log back into the node from the host computer using SSH, using the SSH command shown above.

```
ssh -l brian -p 14603 localhost
```

Testing the network

If everything is working correctly, the virtual PCs and routers in the emulated network are should be able to communicate with every other virtual PC and router in the network.

You may now perform experiments or study the operation of network protocols. For example, you may use the *ping* command to test IP reachability between nodes and you may also look at the routing tables or use *quagga vtysh* commands on the routers to see OSPF protocol status.

For example, use *traceroute* to see that traffic passes through the network between *pc3* and *pc1*:

```
brian@pc3:~$ traceroute 192.168.1.1
traceroute to 192.168.1.1 (192.168.1.1), 30 hops max, 60 byte packets
 1  192.168.3.254 (192.168.3.254)  0.595 ms  0.618 ms  0.589 ms
 2  192.168.101.2 (192.168.101.2)  1.212 ms  1.332 ms  1.195 ms
 3  192.168.1.1 (192.168.1.1)  2.457 ms  2.607 ms  2.400 ms
```

Next Steps

This tutorial worked through the building blocks used to build complex network emulation scenarios. As next steps, you may enable other network protocols in the network topology and study their operation or you may create more complex scenarios using the VirtualBox command line interface.

VirtualBox network lab setup may be automated using popular open-source tools. You may find it beneficial to explore using Vagrant and/or Ansible to automate, manage, and configure VirtualBox network emulations.

Conclusion

I showed how VirtualBox may be used to emulate networks that may be used to study the operation of network protocols and to test networking software. I provided step-by-step instructions for using the VirtualBox graphical user interface to build a network of guest virtual machines that can be managed from the host computer.

Related Posts:

1. **Use Containerlab to emulate open-source routers**
2. **Using VirtualBox linked clones in the GNS3 network simulator**
3. **Vrnetlab: Emulate networks using KVM and Docker**
4. **Installing Debian Linux in a VirtualBox Virtual Machine**
5. **GNS3: Qemu or VirtualBox?**

← PREVIOUS
How to build a network of Linux router...

NEXT →
Psimulator2 forked, updated

27 thoughts on "How to emulate a network using VirtualBox"

Vincent Perrier July 5, 2016 at 11:00 am

Thanks Brian, this post is great to know about network emulation done without the hard work of actually doing it. Your hard work profits to many:) I have 2 questions about virtualbox:
Once a link is created between machines, and when the machines are running, can we erase the link and create another link ?
Do you have to clone the machines or can you have a reference machine root filesystem that is used for all the 5 machines, like the "Copy On Write" that can be done with qcow2 based vm ?

Brian Linkletter July 5, 2016 at 10:00 pm

Hi Vincent,
Thanks for your comment. VirtualBox cannot remove a link when the virtual machines attached to it are still running. VirtualBox supports Linked Clones, which is a Copy on Write system but VirtualBox does not support the automatic creation of virtual machines when they are needed. Each must be prepared in advance.
Thanks,
Brian

Bob November 28, 2016 at 9:28 am

Great tutorial, thanks. But your chart under "network configuration" is right out of whack.

Brian Linkletter November 28, 2016 at 11:27 am

Hi Bob,
Thanks for finding that error. I fixed the table.
Brian

Bob November 29, 2016 at 11:05 am

Cheers Brian, no problem. The ssh port forward method you described worked to a tee. Should the same method also work for a web server on the same vm? (localhost:14600 typed into a browser on the host) I only ask because it's not. Browser window only shows "connecting to 192.168.1.11... (the vm's intnet interface address) and stalls.

Bob November 29, 2016 at 12:24 pm

Sorry, never mind. Virtualbox vm has to be rebooted when port forward changes are made, simple enough and all good. Thanks again.

James Brittain December 20, 2016 at 10:39 am

Hi! Thanks for this, it was very helpful.

I did it with Ubuntu 14.04 since the LFCSA exam still uses the older LTS. When I first set it up nothing worked. Turned out each router was being assigned the same router ID, and I needed to assign it manually by adding a line to the "router ospf" section of ospf.conf. The line "router-id 1.1.1.1," changing 1 to 2 and 3 for the respective routers, made everything work.

Thanks again!
James.

Nathanael Davison December 29, 2016 at 11:56 am

Thanks for the tutorial, very useful.

Do you have any tutorials or can you link any resources demonstrating the use of Vagrant or Ansible (or another tool) for managing a network emulation?

Nathanael

Brian Linkletter January 3, 2017 at 1:49 pm

Hi Nathanael,
No, I have not yet created any tutorials on these topics. I wrote a small survey of available automation tools but I have not written a post showing practical steps required to use them. It's on my list for a topic to cover in the future.
Thank you for your comment!
Brian

bert January 3, 2017 at 11:50 am

Hi Brian, I've been teaching networking mostly to unemployed students for the past 10 years.
Your site is very inspirational. I landed here while thinking of developing a simple python SWITCH emulator. All projects here seem to go beyond these basics, but are very inspirational. Thank You.

Brian Linkletter January 3, 2017 at 1:46 pm

Thank you for your encouragement!

hari January 18, 2017 at 5:06 am

Hi! Nice article. I tried the same example on CentOS 7. After SSH port forwarding ssh not happening saying connected rejected. I checked NAT enabled network enp0s3 settings on CentOS. Its fine but all the VMs have same IP address in this ethernet (10.0.2.15). Does this create a problem?

I found some change in CentOS file like network/interfaces file not available. Can you provide the same example for this OS?

dan January 24, 2017 at 1:26 pm

Very informative… Just one question about Ubuntu server. What parts of it did you include. When building the server you have several options including: ssh open, lamp, etc. Thanks

Brian Linkletter January 24, 2017 at 1:29 pm

Just SSH.

Istvan Toth February 4, 2017 at 8:25 am

Hi Brian,
I have made “How to emulate a network using VirtualBox” topology. It works very well and interesting. This would ask a question: how can one add 5, 6, etc network adapter in command line to a VM? thanks in advance,
Istvan Toth

Omar February 8, 2017 at 3:20 pm

Dear Brian,
This guide helped me a lot in setting up my experimental network. it works perfectly with routing ipv4. Any idea what need to be included for routing ipv6?

Sam Edney February 11, 2017 at 3:26 pm

This is a brilliant tutorial. I just created a 6 node dev network at 35,000ft on the way to Mexico. Wonder if that is a first.

Exactly what I need to test features on my laptop for our production stack.

Thank you very much!

Brian Linkletter February 11, 2017 at 4:02 pm

That's awesome! I'm glad you could use it.

Arham February 28, 2017 at 12:39 pm

Hi Brian,

Thank you for the great tutorial, I have questions regarding your post,
– Is it possible to connect two virtualbox(Windows XP) on router or switch.
– I'll use the two XP as IP Phone but it should be on the same network/subnet.

Sick March 30, 2017 at 3:25 am

this tutorial is very hard for beginner because the texts is copy/paste begin to bash "<<EOF2 … " heuuu excuse me you could have precise the pathfile and write the config file and no you have write a bash script, so please you can edit the post because me too I do not understand.

JIm March 30, 2017 at 12:26 pm

Hi brian,

I completed your tutorial and can preform a traceroute but I am having issues pinging across the network. Any suggestion

Brian Linkletter March 31, 2017 at 12:43 pm

Hi Jum,
This issue would be related to the way you configured either networking or firewall on the nodes in your emulated network. I doubt the VirtualBox infrastructure is the problem.
Brian

Vem April 5, 2017 at 8:36 am

Hi Brain,
Great tutorial, Is it possible to make one of the routers to act as honeypot

Krish April 8, 2017 at 5:48 am

Simply brilliant. Works out of the box. Pure Magic.

In my case I had a barebone ubuntu server built from mini.iso hence I needed to do this in each of the PCs

```
apt-get install traceroute
```

Pingback: Awesome Lab Reference

David C. Snyder April 17, 2017 at 5:14 pm

Great tutorial! I had no significant issues setting things up as described (best to install traceroute in the base image before cloning).

I'm hoping to create something similar that uses BGP instead of OSPF. My understanding of BGP is pretty minimal at this point, so I"m not even sure that this is possible. Thoughts?

Zibi April 22, 2017 at 4:19 am

Hello Brian,

Thanks for your article. Based on this I've built similation on Fedora 25 and Centos 7. Below link:

https://bpm.zciok.blog/2017/04/21/network-simulation-using-virtualbox-and-quagga/

Comments are closed.

Descriptions and evaluations of open-source projects related to network emulation and simulation

Search this blog... Search

List of network simulators and emulators

Network Simulators

cnet

ns-3

OMNet++

Shadow

Network Emulators for Engineers

Cloonix

CORE

IMUNES

EVE-NG

GNS3

Network Emulators for Developers

Containerlab

NetLab

OpenConfig-KNE

vrnetlab

Linux Network Stack Test

Software Defined Networks

Mininet

Containernet

Mobile and Radio Networks

Colosseum

Cooja (Contiki)

CrowNet

CupCarbon

Meshtasticator

NEmu

Network Emulators by Universities

Kathara

Labtainers

VNX and VNUML

Network Demonstrators for High School Students

Educational Network Simulator

CS4G Netsim

Filius

Tools

Do it yourself using Linux tools

TiNet

Copyright © 2024 Open-Source Routing and Network Simulation | Powered by Astra WordPress Theme