RHEL 7 Implementation (Volume 2)

Student Workbook

RHEL 7 IMPLEMENTATION (VOLUME 2)

Red Hat Enterprise Linux 7 RHEL 7 Implementation Volume 2
RHEL 7 Implementation (Volume 2)
Edition 0

Authors: Wander Boessenkool, Chen Chang, Will Dinyes, George Hacker, Rudolf Kastl, Scott McBrien, Douglas Silva, Philip Sweany
Editor: Steven Bonneville

Contributors and Reviewers: Chris Negus, Bruce Wolfe, Rob Locke, Bowe Strickland, Ellen Freeman, Connie Petlitzer, Brandon Nolta, Niraj Nigam, Dirk Davidis, Forrest Taylor, Jim Rigsbee, Ricardo Taniguchi

Adopted for OPENTLC from GLC 299 by Patrick Rutledge

CHAPTER 1

MANAGING IPV6 NETWORKING

Overview	
Goal	To configure and troubleshoot basic IPv6 networking on Red Hat Enterprise Linux systems.
Objectives	• Review how to configure IPv4 networking in RHEL 7. • Explain the basic concepts of IPv6 networking, and read and write condensed IPv6 addresses. • Configure IPv6 networking using command-line tools and configuration files.
Sections	• Review of IPv4 Networking Configuration (and Practice) • IPv6 Networking Concepts (and Practice) • IPv6 Networking Configuration (and Practice)
Lab	• Managing IPv6 Networking

Review of IPv4 Networking Configuration

Objectives

After completing this section, students should be able to configure IPv4 networking using **nmcli** and configuration files in the **/etc/sysconfig/network-scripts** directory.

IPv4 networking

This section assumes that students have a basic understanding of IPv4 networking concepts. In particular, students should know something about IPv4 addresses, network prefixes (and netmasks), default gateways and basic routing, network interfaces, **/etc/hosts**, and name resolution.

NetworkManager overview

In Red Hat Enterprise Linux 7, the configuration of network interfaces is managed by a system daemon called NetworkManager. For NetworkManager:

- A *device* is a network interface.
- A *connection* is a collection of settings that can be configured for a device.
- Only one connection is *active* for any one device at a time. Multiple connections may exist, for use by different devices or to allow a configuration to be altered for the same device.
- Each connection has a *name* or ID that identifies it.
- The persistent configuration for a connection is stored in **/etc/sysconfig/network-scripts/ifcfg-*name***, where *name* is the name of the connection (although spaces are normally replaced with underscores in the file name). This file can be edited by hand if desired.
- The **nmcli** utility can be used to create and edit connection files from the shell prompt.

Viewing networking information

The command **nmcli dev status** will show the status of all network devices:

```
[student@demo ~]$ nmcli dev status
DEVICE  TYPE      STATE         CONNECTION
eno1    ethernet  connected     eno1
eth0    ethernet  connected     static-eth0
eno2    ethernet  disconnected  --
lo      loopback  unmanaged     --
```

The command **nmcli con show** will show a list of all connections. To list only the active connections, add the **--active** option.

```
[student@demo ~]$ nmcli con show
NAME         UUID                                   TYPE            DEVICE
eno2         ff9f7d69-db83-4fed-9f32-939f8b5f81cd  802-3-ethernet  --
static-eth0  72ca57a2-f780-40da-b146-99f71c431e2b  802-3-ethernet  eth0
eno1         87b53c56-1f5d-4a29-a869-8a7bdaf56dfa  802-3-ethernet  eno1
[root@demo ~]# nmcli con show --active
NAME         UUID                                   TYPE            DEVICE
```

```
static-eth0  72ca57a2-f780-40da-b146-99f71c431e2b  802-3-ethernet  eth0
eno1         87b53c56-1f5d-4a29-a869-8a7bdaf56dfa  802-3-ethernet  eno1
```

The **ip addr show** command displays the current configuration of network interfaces on the system. To list only a single interface, add the interface name as the last argument:

```
[student@demo ~]$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,(1)UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
 1000
    (2)link/ether 52:54:00:00:00:0b brd ff:ff:ff:ff:ff:ff
    (3)inet 192.168.0.101/16 brd 192.168.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    (4)inet6 fe80::5054:ff:fe00:b/64 scope link
       valid_lft forever preferred_lft forever
```

1. An active interface is **UP**.
2. The **link/ether** line specifies the hardware (MAC) address of the device.
3. The **inet** line shows an IPv4 address, its network prefix length, and scope.
4. The **inet6** line shows an IPv6 address, its network prefix length, and scope.

Adding a network connection

The **nmcli con add** command is used to add new network connections. The example **nmcli con add** commands that follow assume that the name of the network connection being added is not already in use.

The following command will add a new connection for the interface **eno2**, which will get IPv4 networking information using DHCP and will autoconnect on startup. The configuration will be saved in **/etc/sysconfig/network-scripts/ifcfg-eno2** because the **con-name** is **eno2**.

```
[root@demo ~]# nmcli con add con-name eno2 type ethernet ifname eno2
```

The next example configures the **eno2** interface statically instead, using the IPv4 address and network prefix 192.168.0.5/24 and default gateway 192.168.0.254, but still autoconnects at startup and saves its configuration into the same file. The example is line-wrapped with a shell \ escape.

```
[root@demo ~]# nmcli con add con-name eno2 type ethernet ifname eno2 \
> ip4 192.168.0.5/24 gw4 192.168.0.254
```

Controlling network connections

The **nmcli con up *name*** command will activate the connection *name* on the network interface it is bound to. Note that the command takes the name of a *connection*, not the name of the network interface. Remember that **nmcli con show** can be used to list the names of all available connections.

```
[root@demo ~]# nmcli con up static-eth0
```

The **nmcli dev disconnect *device*** command will disconnect the network interface *device* and bring it down. This command can be abbreviated **nmcli dev dis *device***:

```
[root@demo ~]# nmcli dev dis eth0
```

Important

Use **nmcli dev dis *device*** to deactivate a network interface.

The command **nmcli con down *name*** is normally not the best way to deactivate a network interface. This command will bring down the connection. But by default, most wired system connections are configured with **autoconnect** enabled. This activates the connection as soon as its network interface is available. Since the connection's network interface is still available, **nmcli con down *name*** will bring the interface down, but then NetworkManager will immediately bring it up again unless the connection is entirely disconnected from the interface.

Modifying network connection settings

NetworkManager connections have two kinds of settings. There are *static* connection properties, which are configured by the administrator and stored in the configuration files in **/etc/sysconfig/network-scripts/ifcfg-***. There may also be *active* connection data, which the connection gets from a DHCP server and which are not stored persistently.

To list the current settings for a connection, run the **nmcli con show *name*** command, where *name* is the name of the connection. Settings in lowercase are static properties the administrator can change; settings in all caps are active settings in temporary use for this instance of the connection.

```
[root@demo ~]# nmcli con show static-eth0
connection.id:                          static-eth0
connection.uuid:                        87b53c56-1f5d-4a29-a869-8a7bdaf56dfa
connection.interface-name:              --
connection.type:                        802-3-ethernet
connection.autoconnect:                 yes
connection.timestamp:                   1401803453
connection.read-only:                   no
connection.permissions:
connection.zone:                        --
connection.master:                      --
connection.slave-type:                  --
connection.secondaries:
connection.gateway-ping-timeout:        0
802-3-ethernet.port:                    --
802-3-ethernet.speed:                   0
802-3-ethernet.duplex:                  --
802-3-ethernet.auto-negotiate:          yes
802-3-ethernet.mac-address:             CA:9D:E9:2A:CE:F0
802-3-ethernet.cloned-mac-address:      --
802-3-ethernet.mac-address-blacklist:
802-3-ethernet.mtu:                     auto
802-3-ethernet.s390-subchannels:
802-3-ethernet.s390-nettype:            --
802-3-ethernet.s390-options:
ipv4.method:                            manual
ipv4.dns:                               192.168.0.254
ipv4.dns-search:                        example.com
ipv4.addresses:                         { ip = 192.168.0.2/24, gw = 192.168.0.254 }
ipv4.routes:
```

```
ipv4.ignore-auto-routes:                no
ipv4.ignore-auto-dns:                   no
ipv4.dhcp-client-id:                    --
ipv4.dhcp-send-hostname:                yes
ipv4.dhcp-hostname:                     --
ipv4.never-default:                     no
ipv4.may-fail:                          yes
...
```

The **nmcli con mod *name*** command can be used to change the settings for a connection. These changes will also be saved in the **/etc/sysconfig/network-scripts/ifcfg-*name*** file for the connection. The different settings that are available are documented in the **nm-settings**(5) man page.

To set the IPv4 address to 192.0.2.2/24 and default gateway to 192.0.2.254 for the connection **static-eth0**:

```
[root@demo ~]# nmcli con mod static-eth0 ipv4.addresses "192.0.2.2/24 192.0.2.254"
```

Important

If a connection that got its IPv4 information from a DHCPv4 server is being changed to get it from static configuration files only, the setting **ipv4.method** should also be changed from **auto** to **manual**. Otherwise, the connection may hang or not complete successfully when it is activated, or it may get an IPv4 address from DHCP in addition to the static address.

A number of settings may have multiple values. A specific value can be added to the list or deleted from the list for a setting by adding a **+** or **-** symbol to the start of the setting name.

To add the DNS server 192.0.2.1 to the list of nameservers to use with the connection **static-eth0**:

```
[root@demo ~]# nmcli con mod static-eth0 +ipv4.dns 192.0.2.1
```

By default, changes made with **nmcli con mod *name*** are automatically saved to **/etc/sysconfig/network-scripts/ifcfg-*name***. That file can also be manually edited with a text editor. After doing so, run **nmcli con reload** so that NetworkManager reads the configuration changes.

For backward-compatibility reasons, the directives saved in that file have different names and syntax than the **nm-settings**(5) names. The following table maps some of the key setting names to **ifcfg-*** directives.

Comparison of nm-settings and ifcfg-* Directives

nmcli con mod	ifcfg-* file	Effect
ipv4.method manual	**BOOTPROTO=none**	IPv4 addresses configured statically.
ipv4.method auto	**BOOTPROTO=dhcp**	Will look for configuration settings from a DHCPv4 server. If static addresses are

nmcli con mod	ifcfg-* file	Effect
		also set, will not bring those up until we have information from DHCPv4.
ipv4.addresses "192.0.2.1/24 192.0.2.254"	**IPADDR0=192.0.2.1 PREFIX0=24 GATEWAY0=192.0.2.254**	Sets static IPv4 address, network prefix, and default gateway. If more than one is set for the connection, then instead of **0**, the **ifcfg-*** directives end with 1, 2, 3 and so on.
ipv4.dns 8.8.8.8	**DNS0=8.8.8.8**	Modify **/etc/resolv.conf** to use this **nameserver**.
ipv4.dns-search example.com	**DOMAIN=example.com**	Modify **/etc/resolv.conf** to use this domain in the **search** directive.
ipv4.ignore-auto-dns true	**PEERDNS=no**	Ignore DNS server information from the DHCP server.
connection.autoconnect yes	**ONBOOT=yes**	Automatically activate this connection at boot.
connection.id eth0	**NAME=eth0**	The name of this connection.
connection.interface-name eth0	**DEVICE=eth0**	The connection is bound to the network interface with this name.
802-3-ethernet.mac-address . . .	**HWADDR= . . .**	The connection is bound to the network interface with this MAC address.

Important

Because NetworkManager tends to directly modify the **/etc/resolv.conf** file, direct edits to that file may be overwritten.

To change settings in that file, it is better to set **DNS*n*** and **DOMAIN** directives in the relevant **/etc/sysconfig/network-scripts/ifcfg-*** files.

Deleting a network connection

The **nmcli con del *name*** command will delete the connection named *name* from the system, disconnecting it from the device and removing the file **/etc/sysconfig/network-scripts/ifcfg-*name***.

Modifying the system host name

The **hostname** command displays or temporarily modifies the system's fully qualified host name.

```
[root@demo ~]# hostname
```

```
demo.example.com
```

A static host name may be specified in the **/etc/hostname** file. The **hostnamectl** command is used to modify this file and may be used to view the status of the system's fully qualified host name. If this file does not exist, the host name is set by a reverse DNS query once the interface has an IP address assigned.

```
[root@demo ~]# hostnamectl set-hostname demo.example.com
[root@demo ~]# hostnamectl status
   Static hostname: demo.example.com
         Icon name: computer
           Chassis: n/a
        Machine ID: 9f6fb63045a845d79e5e870b914c61c9
           Boot ID: aa6c3259825e4b8c92bd0f601089ddf7
    Virtualization: kvm
  Operating System: Red Hat Enterprise Linux Server 7.0 (Maipo)
       CPE OS Name: cpe:/o:redhat:enterprise_linux:7.0:GA:server
            Kernel: Linux 3.10.0-121.el7.x86_64
      Architecture: x86_64
[root@demo ~]# cat /etc/hostname
demo.example.com
```

Important

The static host name is stored in **/etc/hostname**. Previous versions of Red Hat Enterprise Linux stored the host name as a variable in the **/etc/sysconfig/network** file.

Summary of commands

The following table is a list of key commands discussed in this section.

Command	Purpose
nmcli dev status	Show the NetworkManager status of all network interfaces.
nmcli con show	List all connections.
nmcli con show *name*	List the current settings for the connection *name*.
nmcli con add con-name *name* ...	Add a new connection named *name*.
nmcli con mod *name* ...	Modify the connection *name*.
nmcli con reload	Tell NetworkManager to reread the configuration files (useful after they have been edited by hand).
nmcli con up *name*	Activate the connection *name*.
nmcli dev dis *dev*	Deactivate and disconnect the current connection on the network interface *dev*.
nmcli con del *name*	Delete the connection *name* and its configuration file.
ip addr show	Show the current network interface address configuration.
hostnamectl set-hostname ...	Persistently set the host name on this system.

References

NetworkManager(8), **nmcli**(1), **nmcli-examples**(5), **nm-settings**(5), **hostnamectl**(1), **resolv.conf**(5), **hostname**(5), **ip**(8), and **ip-address**(8) man pages

IPv6 Networking Concepts

Objectives

After completing this section, students should be able to explain the basic concepts of IPv6 addresses and networking.

IPv6 overview

IPv6 is intended as the replacement for the IPv4 network protocol. The major problem it solves is the exhaustion of IPv4 addresses by using a much larger network address space. It also provides a number of enhancements and new features for network configuration management and support for future protocol changes.

The key reason IPv6 is not yet in wide deployment is that the core protocol does not have a simple way for systems that only have IPv6 addresses to communicate with systems that only have IPv4 addresses.

The best transition plan at present is to provide all hosts with both IPv4 and IPv6 addresses, so that Internet resources only using one of the protocols can be reached from the host. This is called a *dual-stack* configuration, and is the approach on which this course will focus.

Note

There are a number of promising transition methods in development to allow IPv6-only hosts to use the IPv4 Internet or support other forms of IPv4/IPv6 translation, such as NAT64 (RFC 6145) and 464XLAT (RFC 6877), but they are beyond the scope of this course.

The basic position of the Internet Engineering Task Force (IETF) is that network operators using IPv4 should “obtain an IPv6 prefix, turn on IPv6 routing within their networks and between themselves and any peer, upstream, or downstream neighbors, enable it on their computers, and use it in normal processing. This should be done while leaving IPv4 stable, until a point is reached that any communication that can be carried out could use either protocol equally well. At that point, the economic justification for running both becomes debatable, and network operators can justifiably turn IPv4 off.” (RFC 6144, Introduction)

Interpreting IPv6 addresses

IPv6 addresses

An IPv6 address is a 128-bit number, normally expressed as eight colon-separated groups of four hexadecimal nibbles (half-bytes). Each nibble represents four bits of the IPv6 address, so each group represents 16 bits of the IPv6 address.

```
2001:0db8:0000:0010:0000:0000:0000:0001
```

To make it easier to write IPv6 addresses, leading zeros in a colon-separated group do not need to be written. However, at least one nibble must be written in each field. Zeros which follow a nonzero nibble in the group do need to be written.

```
2001:db8:0:10:0:0:0:1
```

Since addresses with long strings of zeros are common, one or more groups of consecutive zeros may be combined with *exactly one* **::** block.

```
2001:db8:0:10::1
```

Notice that under these rules, **2001:db8::0010:0:0:0:1** would be another less convenient way to write the example address. But it is a valid representation of the same address, and this can confuse administrators new to IPv6. Some tips for writing consistently readable addresses:

1. Leading zeros in a group must always be suppressed.
2. Use **::** to shorten as much as possible. If two runs of zeros are equal in length, shorten the leftmost run of zeros by preference.
3. Although it is allowed, do not use **::** to shorten one group of zeros. Use **:0:** instead, and save **::** for runs of zeros longer than a single group.
4. Always use lowercase letters for hexadecimal numbers **a** through **f**.

Important

When including a TCP or UDP network port after an IPv6 address, always enclose the IPv6 address in square brackets so that the port does not look like it is part of the address.

```
[2001:db8:0:10::1]:80
```

IPv6 subnets

A normal unicast address is divided into two parts: the *network prefix* and *interface ID*. The network prefix identifies the subnet. No two network interfaces on the same subnet can have the same interface ID; the interface ID identifies a particular interface on the subnet.

Unlike IPv4, IPv6 has a standard subnet mask, which is used for almost all normal addresses, **/64**. In this case, half of the address is the network prefix and half of it is the interface ID. This means that a single subnet can hold as many hosts as necessary.

Typically, the network provider will allocate a shorter prefix to an organization, such as a **/48**. This leaves the rest of the network part for assigning subnets from that allocated prefix. For a **/48** allocation, that leaves 16 bits for subnets (up to 65536 subnets).

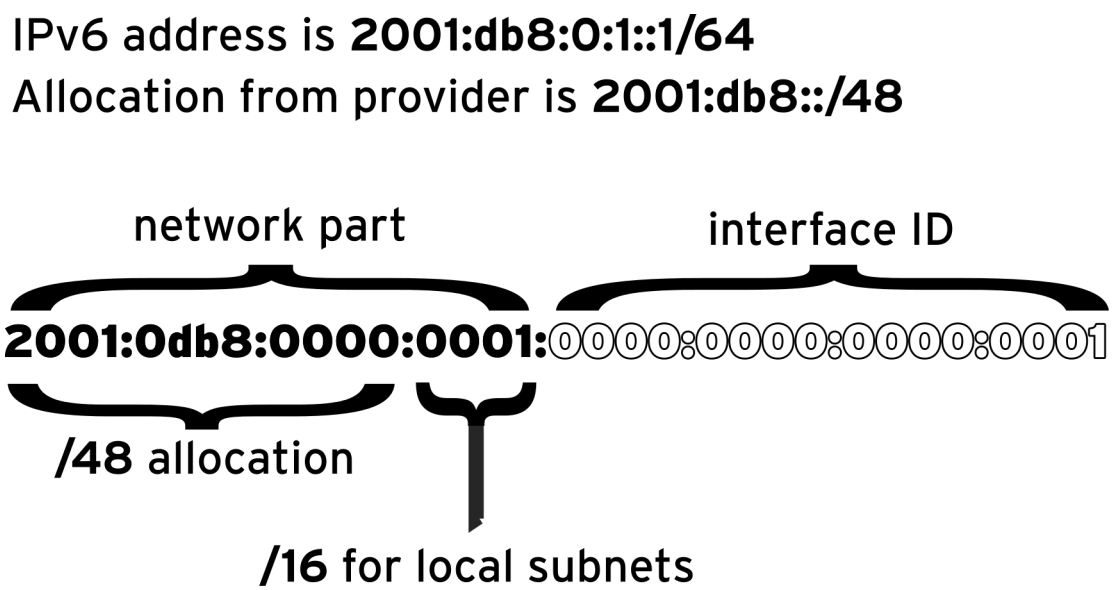

Figure 1.1: IPv6 address parts and subnetting

IPv6 address allocation

Common IPv6 Addresses and Networks

IPv6 address or network	Purpose	Description
::1/128	localhost	The IPv6 equivalent to **127.0.0.1/8**, set on the loopback interface.
::	The unspecified address	The IPv6 equivalent to **0.0.0.0**. For a network service, this could indicate that it is listening on all configured IP addresses.
::/0	The default route (the IPv6 Internet)	The IPv6 equivalent to **0.0.0.0/0**. The default route in the routing table matches this network; the router for this network is where all traffic is sent for which there is not a better route.
2000::/3	Global unicast addresses	"Normal" IPv6 addresses are currently being allocated from this space by IANA. This is equivalent to all the networks ranging from **2000::/16** through **3fff::/16**.
fd00::/8	Unique local addresses (RFC 4193)	IPv6 has no direct equivalent of RFC 1918 private address space, although this is close. A site can use these to self-allocate a private routable IP address space inside the organization, but these networks cannot be used on the global Internet. The site must *randomly* select a **/48** from this space, but it can subnet the allocation into **/64** networks normally.

IPv6 address or network	Purpose	Description
fe80::/64	Link-local addresses	Every IPv6 interface automatically configures a *link-local* address that only works on the local link on this network. This will be discussed in more detail later.
ff00::/8	Multicast	The IPv6 equivalent to **224.0.0.0/4**. Multicast is used to transmit to multiple hosts at the same time, and is particularly important in IPv6 because it has no broadcast addresses.

Link-local addresses

A *link-local address* in IPv6 is an unroutable address which is used only to talk to hosts on a specific network link. Every network interface on the system is automatically configured with a link-local address on the **fe80::** network. To ensure that it is unique, the interface ID of the link-local address is constructed from the network interface's Ethernet hardware address. The usual procedure to convert the 48-bit MAC address to a 64-bit interface ID is to set bit 7 of the MAC address and insert **ff:fe** between its two middle bytes.

- Network prefix: **fe80::/64**
- MAC address: **00:11:22:aa:bb:cc**
- Link-local address: **fe80::211:22ff:feaa:bbcc/64**

The link-local addresses of other machines can be used like normal addresses by other hosts on the same link. Since *every* link has a **fe80::/64** network on it, the routing table cannot be used to select the outbound interface correctly. The link to use when talking to a link-local address must be specified with a *scope identifier* at the end of the address. The scope identifier consists of **%** followed by the name of the network interface.

For example, to use **ping6** to ping the link-local address **fe80::211:22ff:feaa:bbcc** using the link connected to the **eth0** network interface, the correct command would be:

```
[student@demo ~]$ ping6 fe80::211:22ff:feaa:bbcc%eth0
```

Note

Scope identifiers are only needed when contacting addresses that have "link" scope. Normal global addresses are used just like they are in IPv4, and select their outbound interfaces from the routing table.

Multicast

Multicast plays a larger role in IPv6 than in IPv4 because there is no broadcast address in IPv6. One key multicast address in IPv6 is **ff02::1**, the all-nodes link-local address. Pinging this address will send traffic to all nodes on the link. Link-scope multicast addresses (starting **ff02::/8**) need to be specified with a scope identifier, just like a link-local address.

```
[student@demo ~]$ ping6 ff02::1%eth0
```

```
PING ff02::1%eth0(ff02::1) 56 data bytes
64 bytes from fe80::211:22ff:feaa:bbcc: icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from fe80::200:aaff:fe33:2211: icmp_seq=1 ttl=64 time=102 ms (DUP!)
64 bytes from fe80::bcd:efff:fea1:b2c3: icmp_seq=1 ttl=64 time=103 ms (DUP!)
64 bytes from fe80::211:22ff:feaa:bbcc: icmp_seq=2 ttl=64 time=0.079 ms
...
```

IPv6 address configuration

IPv4 has two ways in which addresses get configured on network interfaces. Network addresses may be configured on interfaces manually by the administrator, or dynamically from the network using DHCP. IPv6 also supports manual configuration, and two methods of dynamic configuration, one of which is DHCPv6.

Static addressing

Interface IDs for static IPv6 addresses can be selected at will, just like IPv4. In IPv4, there were two addresses on a network that could not be used, the lowest address in the subnet and the highest address in the subnet. In IPv6, the following interface IDs are reserved and cannot be used for a normal network address on a host.

- The all-zeros identifier **`0000:0000:0000:0000`** ("subnet router anycast") used by all routers on the link. (For the **`2001:db8::/64`** network, this would be the address **`2001:db8::`**.)
- The identifiers **`fdff:ffff:ffff:ff80`** through **`fdff:ffff:ffff:ffff`**.

DHCPv6 configuration

DHCPv6 works a little differently than DHCP for IPv4, because there is no broadcast address. Essentially, a host sends a DHCPv6 request from its link-local address to port 547/UDP on **`ff02::1:2`**, the all-dhcp-servers link-local multicast group. The DHCPv6 server then usually sends a reply with appropriate information to port 546/UDP on the client's link-local address.

The *dhcp* package in RHEL 7 provides support for a DHCPv6 server.

SLAAC configuration

In addition to DHCPv6, IPv6 also supports a second dynamic configuration method, called *Stateless Address Autoconfiguration* (SLAAC). Using SLAAC, the host brings up its interface with a link-local **`fe80::/64`** address normally. It then sends a "router solicitation" to **`ff02::2`**, the all-routers link-local multicast group. An IPv6 router on the local link responds to the host's link-local address with a network prefix and possibly other information. The host then uses that network prefix with an interface ID that it normally constructs in the same way that link-local addresses are constructed. The router periodically sends multicast updates ("router advertisements") to confirm or update the information it provided.

The *radvd* package in RHEL 7 allows a RHEL-based IPv6 router to provide SLAAC through router advertisements.

Important

A typical RHEL 7 machine configured to get IPv4 addresses through DHCP is usually also configured to use SLAAC to get IPv6 addresses. This can result in machines unexpectedly obtaining IPv6 addresses when an IPv6 router is added to the network.

Some IPv6 deployments combine SLAAC and DHCPv6, using SLAAC to only provide network address information and DHCPv6 to provide other information, such as which DNS servers and search domains to configure.

References

`ping`(8), **`radvd`**(8), and **`dhcpd`**(8) man pages

Selected IETF RFC references:

> RFC 2460: Internet Protocol, Version 6 (IPv6) Specification
> http://tools.ietf.org/html/rfc2460

> RFC 4291: IP Version 6 Addressing Architecture
> http://tools.ietf.org/html/rfc4291

> RFC 5952: A Recommendation For IPv6 Address Text Representation
> http://tools.ietf.org/html/rfc5952

> RFC 4862: IPv6 Stateless Address Autoconfiguration
> http://tools.ietf.org/html/rfc4862

> RFC 3315: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
> http://tools.ietf.org/html/rfc3315

> RFC 3736: Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6
> http://tools.ietf.org/html/rfc3736

> RFC 4193: Unique Local IPv6 Unicast Addresses
> http://tools.ietf.org/html/rfc4193

IPv6 Networking Configuration

Objectives

After completing this section, students should be able to configure IPv6 networking using **nmcli** and configuration files in the **/etc/sysconfig/network-scripts** directory.

NetworkManager and IPv6

To work with IPv6 addresses using NetworkManager, all the commands that are used with IPv4 networking work with IPv6 networking. There are some different settings that are relevant for connections, but most commands will be similar for IPv6 configuration.

Adding an IPv6 network connection

The **nmcli con add** command is used to add new network connections.

The following command, shown in a previous section of this chapter, will add a new connection for the interface **eno2**, which will autoconnect at startup, getting IPv4 networking information using DHCPv4. It will *also* get IPv6 networking settings by listening for router advertisements on the local link.

```
[root@demo ~]# nmcli con add con-name eno2 type ethernet ifname eno2
```

The next example configures the **eno2** interface statically instead, using the IPv6 address and network prefix 2001:db8:0:1::c000:207/64 and default IPv6 gateway 2001:db8:0:1::1, and the IPv4 address and network prefix 192.0.2.7/24 and default IPv4 gateway 192.0.2.1, but still autoconnects at startup and saves its configuration into **/etc/sysconfig/network-scripts/ifcfg-eno2**. The example is line-wrapped with a shell **** escape.

```
[root@demo ~]# nmcli con add con-name eno2 type ethernet ifname eno2 \
> ip6 2001:db8:0:1::c000:207/64 gw6 2001:db8:0:1::1 ip4 192.0.2.7/24 gw4 192.0.2.1
```

Modifying network connection settings for IPv6

The **nmcli con show *name*** command, where *name* is the name of the connection, can be used to view IPv6-related settings:

```
[root@demo ~]# nmcli con show static-eth0 | grep ipv6
ipv6.method:                    manual
ipv6.dns:                       2001:4860:4860::8888
ipv6.dns-search:                example.com
ipv6.addresses:                 { ip = 2001:db8:0:1::7/64, gw = 2001:db8:0:1::1 }
ipv6.routes:
ipv6.ignore-auto-routes:        no
ipv6.ignore-auto-dns:           no
ipv6.never-default:             no
ipv6.may-fail:                  yes
ipv6.ip6-privacy:               -1 (unknown)
ipv6.dhcp-hostname:             --
[root@demo ~]#
```

Likewise, **nmcli con mod *name*** can be used to adjust how connections set IPv6 addresses.

To set the IPv6 address to 2001:db8:0:1::a00:1/64 and default gateway to 2001:db8:0:1::1 for the connection **static-eth0**:

```
[root@demo ~]# nmcli con mod static-eth0 ipv6.address "2001:db8:0:1::a00:1/64
 2001:db8:0:1::1"
```

Important

If a connection that got its IPv6 information by SLAAC or a DHCPv6 server is being changed to get it from static configuration files only, the setting **ipv6.method** should also be changed from **auto** or **dhcp** to **manual**. Otherwise, the connection may hang or not complete successfully when it is activated, or it may get an IPv6 address from SLAAC or DHCPv6 in addition to the static address.

A number of settings may have multiple values. A specific value can be added to the list or deleted from the list for a setting by adding a **+** or **-** symbol to the start of the setting name.

To add the DNS server 2001:4860:4860::8888 to the list of nameservers to use with the connection **static-eth0**:

```
[root@demo ~]# nmcli con mod static-eth0 +ipv6.dns 2001:4860:4860::8888
```

Note

Static IPv4 and IPv6 DNS settings all end up as **nameserver** directives in **/etc/resolv.conf**. It may be a good idea to ensure that there is, at minimum, an IPv4-reachable nameserver (assuming a dual-stack system) and preferably at least one nameserver using each protocol in case of connectivity issues with either IPv4 or IPv6 networking.

Remember that the file **/etc/sysconfig/network-scripts/ifcfg-*name*** can be directly edited, and that **nmcli con reload** must be run after saving so that NetworkManager reads the configuration changes.

The following table maps some of the key NetworkManager setting names relevant to IPv6 connections to **ifcfg-*** directives.

Comparison of nm-settings and ifcfg-* Directives

nmcli con mod	ifcfg-* file	Effect
ipv6.method manual	**IPV6_AUTOCONF=no**	IPv6 addresses configured statically.
ipv6.method auto	**IPV6_AUTOCONF=yes**	Will configure network settings using SLAAC from router advertisements.
ipv6.method dhcp	**IPV6_AUTOCONF=no** **DHCPV6C=yes**	Will configure network settings by using DHCPv6, but not SLAAC.

nmcli con mod	ifcfg-* file	Effect
ipv6.addresses "2001:db8::a/64 2001:db8::1"	**IPV6ADDR=2001:db8::a/64** **IPV6_DEFAULTGW=2001:db8::1**	Sets static IPv4 address, network prefix, and default gateway. If more than one address is set for the connection, **IPV6_SECONDARIES** takes a double-quoted list of space-delimited address/prefix definitions.
ipv6.dns . . .	**DNS0= . . .**	Modify **/etc/resolv.conf** to use this **nameserver**. Exactly the same as IPv4.
ipv6.dns-search example.com	**DOMAIN=example.com**	Modify **/etc/resolv.conf** to use this domain in the **search** directive. Exactly the same as IPv4.
ipv6.ignore-auto-dns true	**IPV6_PEERDNS=no**	Ignore DNS server information from the DHCP server.
connection.autoconnect yes	**ONBOOT=yes**	Automatically activate this connection at boot.
connection.id eth0	**NAME=eth0**	The name of this connection.
connection.interface-name eth0	**DEVICE=eth0**	The connection is bound to the network interface with this name.
802-3-ethernet.mac-address . . .	**HWADDR= . . .**	The connection is bound to the network interface with this MAC address.

Viewing IPv6 networking information

Both **nmcli dev status** to show the NetworkManager status of all network devices and **nmcli con show** to show the list of available connections work exactly as they do for IPv4.

The **ip addr show** command still displays the current configuration of network interfaces on the system. The example that follows calls out some items relevant to IPv6.

```
[student@demo ~]$ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,(1)UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
 1000
    (2)link/ether 52:54:00:00:00:0b brd ff:ff:ff:ff:ff:ff
    (3)inet 192.0.2.2/24 brd 192.0.2.255 scope global eth0
       valid_lft forever preferred_lft forever
    (4)inet6 2001:db8:0:1:5054:ff:fe00:b/64 scope global
       valid_lft forever preferred_lft forever
    (5)inet6 fe80::5054:ff:fe00:b/64 scope link
       valid_lft forever preferred_lft forever
```

1. An active interface is **UP**.
2. The **link/ether** line specifies the hardware (MAC) address of the device.
3. The **inet** line shows an IPv4 address, its network prefix length, and scope.
4. The **inet6** line shows an IPv6 address, its network prefix length, and scope. This address is of *global* scope and is normally used.
5. This **inet6** line is for an address of *link* scope and can only be used for communication on the local Ethernet link.

The **ip -6 route show** command displays the IPv6 routing table for the system:

```
[root@demo ~]# ip -6 route show
unreachable ::/96 dev lo  metric 1024  error -101
unreachable ::ffff:0.0.0.0/96 dev lo  metric 1024  error -101
2001:db8:0:1::/64 dev eth0  proto kernel  metric 256
unreachable 2002:a00::/24 dev lo  metric 1024  error -101
unreachable 2002:7f00::/24 dev lo  metric 1024  error -101
unreachable 2002:a9fe::/32 dev lo  metric 1024  error -101
unreachable 2002:ac10::/28 dev lo  metric 1024  error -101
unreachable 2002:c0a8::/32 dev lo  metric 1024  error -101
unreachable 2002:e000::/19 dev lo  metric 1024  error -101
unreachable 3ffe:ffff::/32 dev lo  metric 1024  error -101
fe80::/64 dev eth0  proto kernel  metric 256
default via 2001:db8:0:1::ffff dev eth0  proto static  metric 1024
```

In the previous example, ignore the **unreachable** routes, which point at networks which are never to be used. That leaves three routes:

1. To the 2001:db8:0:1::/64 network using the eth0 interface (which presumably has an address on that network).

2. To the fe80::/64 network using the eth0 interface, for the link-local address. On a system with multiple interfaces, there will be a route to fe80::/64 out each interface for each link-local address.

3. A default route to all networks on the IPv6 Internet (the ::/0 network) that don't have a more specific route on the system, through the router at 2001:db8:0:1::ffff, reachable with the eth0 device.

IPv6 troubleshooting tools

Connectivity

The **ping6** command is the IPv6 version of **ping** in Red Hat Enterprise Linux. It communicates over IPv6 and can take IPv6 addresses, but otherwise works like **ping**.

```
[root@demo ~]# ping6 2001:db8:0:1::1
PING 2001:db8:0:1::1(2001:db8:0:1::1) 56 data bytes
64 bytes from 2001:db8:0:1::1: icmp_seq=1 ttl=64 time=18.4 ms
64 bytes from 2001:db8:0:1::1: icmp_seq=2 ttl=64 time=0.178 ms
64 bytes from 2001:db8:0:1::1: icmp_seq=3 ttl=64 time=0.180 ms
^C
--- 2001:db8:0:1::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.178/6.272/18.458/8.616 ms
[root@demo ~]#
```

Link-local addresses and the link-local all-nodes multicast group (ff02::1) can be pinged, but the network interface to use must be specified explicitly with a scope zone identifier (such as ff02::1*%eth0*). If this is left out, the error **`connect: Invalid argument`** will be displayed.

Pinging ff02::1 can be useful for finding other IPv6 nodes on the local network.

```
[root@rhel7 ~]# ping6 ff02::1%eth1
PING ff02::1%eth1(ff02::1) 56 data bytes
64 bytes from fe80::78cf:7fff:fed2:f97b: icmp_seq=1 ttl=64 time=22.7 ms
64 bytes from fe80::f482:dbff:fe25:6a9f: icmp_seq=1 ttl=64 time=30.1 ms (DUP!)
64 bytes from fe80::78cf:7fff:fed2:f97b: icmp_seq=2 ttl=64 time=0.183 ms
64 bytes from fe80::f482:dbff:fe25:6a9f: icmp_seq=2 ttl=64 time=0.231 ms (DUP!)
^C
--- ff02::1%eth1 ping statistics ---
2 packets transmitted, 2 received, +2 duplicates, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.183/13.320/30.158/13.374 ms
[root@rhel7 ~]# ping6 -c 1 fe80::f482:dbff:fe25:6a9f%eth1
PING fe80::f482:dbff:fe25:6a9f%eth1(fe80::f482:dbff:fe25:6a9f) 56 data bytes
64 bytes from fe80::f482:dbff:fe25:6a9f: icmp_seq=1 ttl=64 time=22.9 ms

--- fe80::f482:dbff:fe25:6a9f%eth1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 22.903/22.903/22.903/0.000 ms
```

Remember that IPv6 link-local addresses can be used by other hosts on the same link, just like normal addresses.

```
[student@demo ~]$ ssh fe80::f482:dbff:fe25:6a9f%eth1
student@fe80::f482:dbff:fe25:6a9f%eth1's password:
Last login: Thu Jun  5 15:20:10 2014 from demo.example.com
[student@server ~]$
```

Routing

The **`tracepath6`** and **`traceroute -6`** commands are the equivalent to **`tracepath`** and **`traceroute`** for IPv6.

```
[root@demo ~]# tracepath6 2001:db8:0:2::451
 1?: [LOCALHOST]                      0.091ms pmtu 1500
 1:  2001:db8:0:1::ba                 0.214ms
 2:  2001:db8:0:1::1                  0.512ms
 3:  2001:db8:0:2::451   0.559ms reached
     Resume: pmtu 1500 hops 3 back 3
```

Ports and services

Either the **`ss`** command or the **`netstat`** command can display information about network sockets, and they take almost identical options.

```
[root@demo ~]# ss -A inet -n
Netid State      Recv-Q Send-Q       Local Address:Port          Peer Address:Port
tcp   ESTAB      0      0          192.168.122.98:22           192.168.122.1:35279
tcp   ESTAB      0      0         2001:db8:0:1::ba:22         2001:db8:0:1::1:40810
[root@demo ~]# netstat -46n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address        Foreign Address        State
tcp        0      0 192.168.122.98:22    192.168.122.1:35279    ESTABLISHED
tcp6       0      0 2001:db8:0:1::ba:22  2001:db8:0:1::1:40810  ESTABLISHED
```

Options for **ss** and **netstat**

Option	Description
-n	Show numbers instead of names for interfaces and ports.
-t	Show TCP sockets.
-u	Show UDP sockets.
-l	Show only listening sockets.
-a	Show all (listening and established) sockets.
-p	Show the process using the sockets.
-A inet	Display active connections (but not listening sockets) for the **inet** address family. That is, ignore local UNIX domain sockets. For **ss**, both IPv4 and IPv6 connections will be displayed. For **netstat**, only IPv4 connections will be displayed. (**netstat -A inet6** will display IPv6 connections, and **netstat -46** will display IPv4 and IPv6 at the same time.)

References

NetworkManager(8), **nmcli**(1), **nmcli-examples**(5), **nm-settings**(5), **ip**(8), **ip-address**(8), **ip-route**(8), **ping6**(8), **tracepath6**(8), **traceroute**(8), **ss**(8), and **netstat**(8) man pages

Practice: Configuring IPv6 Networking

Guided exercise

In this lab, you will configure a network interface with a static IPv6 address. Once the interface is configured, you will confirm that it works and identify other IPv6 nodes on the local network. You will also explore the contents of the configuration file created by NetworkManager.

Resources:	
Files:	**/etc/sysconfig/network-scripts/ifcfg-eth1**
Machines:	**server1**

Outcomes:
The **eth1** network interface on your **server1** machine will be managed by NetworkManager with a connection named **eth1**. It will statically configure an IPv6 address of fddb:fe2a:ab1e::c0a8:1/64 and use fe80::2ec2:60ff:fe10:3213/64 as the IPv6 gateway.

Before you begin...

- Reset the server1 system.
- Become the **root** user.

1. Before making any changes, display the list of existing network interfaces in order to determine the system's current configuration. Also determine which interfaces are managed by NetworkManager.

   1.1. The **ip link** command will display all of the network interfaces recognized by the system.

```
[root@server1 ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
 DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 mode DEFAULT qlen 1000
    link/ether 52:54:00:00:07:0b brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 mode DEFAULT qlen 1000
    link/ether ce:c4:7c:28:4c:7a brd ff:ff:ff:ff:ff:ff
```

   1.2. Use the **nmcli** to list the network interfaces that NetworkManager manages.

```
[root@server1 ~]# nmcli con show
NAME         UUID                                  TYPE            DEVICE
System eth0  5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03  802-3-ethernet  eth0
```

2. Create a NetworkManager connection, called **eth1**, for the **eth1** network interface. Redisplay the list of managed interfaces to confirm NetworkManager manages **eth1**.

   2.1. Use **nmcli** to create the connection for **eth1**.

```
[root@server1 ~]# nmcli con add con-name eth1 type ethernet ifname eth1
```

```
Connection 'eth1' (0d687259-c64b-4e5b-bece-cabbe952e46f) successfully added.
```

2.2. Display the new list of interfaces managed by NetworkManager. **eth1** should be somewhere in the list.

```
[root@server1 ~]# nmcli con show
NAME         UUID                                  TYPE            DEVICE
eth1         0d687259-c64b-4e5b-bece-cabbe952e46f  802-3-ethernet  eth1
System eth0  5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03  802-3-ethernet  eth0
```

3. Display the current IP address information for **eth1**.

```
[root@server1 ~]# ip addr show eth1
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen
 1000
    link/ether 06:8f:6e:13:6e:8e brd ff:ff:ff:ff:ff:ff
    inet6 fe80::48f:6eff:fe13:6e8e/64 scope link
       valid_lft forever preferred_lft forever
```

It will have an IPv6 link-local address assigned to it (the address on the **fe80::/64** network).

4. Display the initial, default NetworkManager IPv6 configuration settings for the connection.

```
[root@server1 ~]# nmcli con show eth1 | grep ipv6
ipv6.method:                            auto
ipv6.dns:
ipv6.dns-search:
ipv6.addresses:
ipv6.routes:
ipv6.ignore-auto-routes:                no
ipv6.ignore-auto-dns:                   no
ipv6.never-default:                     no
ipv6.may-fail:                          yes
ipv6.ip6-privacy:                       -1 (unknown)
ipv6.dhcp-hostname:                     --
```

5. Configure **eth1** to have a static IPv6 address of **fddb:fe2a:ab1e::c0a8:1** with a standard **/64** subnet prefix. Use **fe80::2ec2:60ff:fe10:3213** as the IPv6 gateway.

```
[root@server1 ~]# nmcli con mod eth1 ipv6.addresses 'fddb:fe2a:ab1e::c0a8:1/64
 fe80::2ec2:60ff:fe10:3213'
[root@server1 ~]# nmcli con mod eth1 ipv6.method manual
```

6. Restart the **eth1** network interface and confirm its new IPv6 address configuration.

6.1. Bounce the **eth1** interface by taking it down, then bringing it back up.

```
[root@server1 ~]# nmcli con down eth1
[root@server1 ~]# nmcli con up eth1
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/2)
```

6.2. Use the **ip addr** command to confirm the interface's configuration.

```
[root@server1 ~]# ip addr show dev eth1
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 qlen 1000
    link/ether 06:8f:6e:13:6e:8e brd ff:ff:ff:ff:ff:ff
    inet6 fddb:fe2a:ab1e::c0a8:1/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::48f:6eff:fe13:6e8e/64 scope link
       valid_lft forever preferred_lft forever
```

Notice the global address, fddb:fe2a:ab1e::c0a8:1/64, is available for use.

7. Ping **eth1**'s own IPv6 address.

```
[root@server1 ~]# ping6 fddb:fe2a:ab1e::c0a8:1
PING fddb:fe2a:ab1e::c0a8:1(fddb:fe2a:ab1e::c0a8:1) 56 data bytes
64 bytes from fddb:fe2a:ab1e::c0a8:1: icmp_seq=1 ttl=64 time=0.141 ms
64 bytes from fddb:fe2a:ab1e::c0a8:1: icmp_seq=2 ttl=64 time=0.081 ms
^C
--- fddb:fe2a:ab1e::c0a8:1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.081/0.111/0.141/0.030 ms
```

8. Ping the IPv6 gateway to make sure it is reachable by **eth1**.

```
[root@server1 ~]# ping6 fe80::2ec2:60ff:fe10:3213%eth1
PING fe80::2ec2:60ff:fe10:3213(fe80::2ec2:60ff:fe10:3213) 56 data bytes
64 bytes from fe80::2ec2:60ff:fe10:3213: icmp_seq=1 ttl=64 time=0.254 ms
64 bytes from fe80::2ec2:60ff:fe10:3213: icmp_seq=2 ttl=64 time=0.123 ms
64 bytes from fe80::2ec2:60ff:fe10:3213: icmp_seq=3 ttl=64 time=0.119 ms
64 bytes from fe80::2ec2:60ff:fe10:3213: icmp_seq=4 ttl=64 time=0.123 ms
64 bytes from fe80::2ec2:60ff:fe10:3213: icmp_seq=5 ttl=64 time=0.090 ms
^C
--- fe80::2ec2:60ff:fe10:3213 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.090/0.141/0.254/0.059 ms
```

9. Use the **ip** command to display the IPv6 routing table. Note the default gateway listed for IPv6.

```
[root@server1 ~]# ip -6 route
unreachable ::/96 dev lo  metric 1024  error -101
unreachable ::ffff:0.0.0.0/96 dev lo  metric 1024  error -101
unreachable 2002:a00::/24 dev lo  metric 1024  error -101
unreachable 2002:7f00::/24 dev lo  metric 1024  error -101
unreachable 2002:a9fe::/32 dev lo  metric 1024  error -101
unreachable 2002:ac10::/28 dev lo  metric 1024  error -101
unreachable 2002:c0a8::/32 dev lo  metric 1024  error -101
unreachable 2002:e000::/19 dev lo  metric 1024  error -101
unreachable 3ffe:ffff::/32 dev lo  metric 1024  error -101
fddb:fe2a:ab1e::/64 dev eth1  proto kernel  metric 256
fe80::/64 dev eth0  proto kernel  metric 256
fe80::/64 dev eth1  proto kernel  metric 256
default via fe80::2ec2:60ff:fe10:3213 dev eth1  proto static  metric 1024
```

10. Discover other local IPv6 nodes on the network. Ping the link-local all-nodes multicast group (**ff02::1**) through the **eth1** interface to see what other hosts respond.

```
[root@server1 ~]# ping6 ff02::1%eth1
PING ff02::1%eth1(ff02::1) 56 data bytes
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=1 ttl=64 time=0.298 ms
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=1 ttl=64 time=0.306 ms (DUP!)
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=2 ttl=64 time=0.161 ms (DUP!)
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=3 ttl=64 time=0.107 ms
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=3 ttl=64 time=0.136 ms (DUP!)
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=4 ttl=64 time=0.111 ms
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=4 ttl=64 time=0.143 ms (DUP!)
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=5 ttl=64 time=0.131 ms
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=5 ttl=64 time=0.167 ms (DUP!)
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=6 ttl=64 time=0.109 ms
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=6 ttl=64 time=0.141 ms (DUP!)
64 bytes from fe80::707e:68ff:fe3e:fd23: icmp_seq=7 ttl=64 time=0.116 ms
64 bytes from fe80::fc46:acff:fefe:10b7: icmp_seq=7 ttl=64 time=0.150 ms (DUP!)
^C
--- ff02::1%eth1 ping statistics ---
7 packets transmitted, 7 received, +7 duplicates, 0% packet loss, time 5999ms
rtt min/avg/max/mdev = 0.107/0.157/0.306/0.062 ms
```

11. Identify the interface configuration file for the **eth1** network interface in **/etc/sysconfig/network-scripts**. View the file contents and note which variable assignments relate to the IPv6 configuration that was performed earlier.

```
[root@server1 ~]# ls /etc/sysconfig/network-scripts/ifcfg-*
/etc/sysconfig/network-scripts/ifcfg-eth1
/etc/sysconfig/network-scripts/ifcfg-eth0
/etc/sysconfig/network-scripts/ifcfg-lo
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth1
TYPE=Ethernet
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=no
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
NAME=eth1
UUID=4214d89b-f409-4853-8e31-4e673845e1a1
DEVICE=eth1
ONBOOT=yes
IPV6ADDR=fddb:fe2a:ab1e::c0a8:1/64
IPV6_DEFAULTGW=fe80::2ec2:60ff:fe10:3213
PEERDNS=yes
PEERROUTES=yes
```

Summary

Review of IPv4 Networking Configuration

In this section, students learned how to:

- Configure IPv4 networking using **`nmcli`**.
- Configure IPv4 networking using **`/etc/sysconfig/network-scripts/ifcfg-*`** files.

IPv6 Networking Concepts

In this section, students learned how to:

- Explain the basic details of IPv6 networking.
- Interpret text representations of IPv6 addresses.

IPv6 Networking Configuration

In this section, students learned how to:

- Configure IPv6 networking using **`nmcli`**.
- Configure IPv6 networking using **`/etc/sysconfig/network-scripts/ifcfg-*`** files.

CHAPTER 2

CONFIGURING LINK AGGREGATION AND BRIDGING

Overview	
Goal	To configure and troubleshoot advanced network interface functionality, including teaming and local software bridges.
Objectives	• Use network teaming to provide link redundancy or higher throughput. • Manage a network team interface. • Manage local software bridges and associated interfaces.
Sections	• Configuring Network Teaming (and Practice) • Managing Network Teaming (and Practice) • Configuring Software Bridges (and Practice)
Lab	• Configuring Link Aggregation and Bridging

Configuring Network Teaming

Objectives

After completing this section, students should be able to combine two network links using network teaming to provide link redundancy or higher throughput.

Network teaming

Network teaming is method for linking NICs together logically to allow for failover or higher throughput. Teaming is a new implementation that does not affect the older bonding driver in the Linux kernel; it offers an alternate implementation. Red Hat Enterprise Linux 7 supports channel bonding for backward compatability. Network teaming provides better performance and is more extensible because of its modular design.

Red Hat Enterprise Linux 7 implements network teaming with a small kernel driver and a user-space daemon, **`teamd`**. The kernel handles network packets efficiently and **`teamd`** handles logic and interface processing. Software, called *runners*, implement load balancing and active-backup logic, such as **`roundrobin`**. The following runners are available to **`teamd`**:

- **`broadcast`**: a simple runner which transmits each packet from all ports.
- **`roundrobin`**: a simple runner which transmits packets in a round-robin fashing from each of the ports.
- **`activebackup`**: this is a failover runner which watches for link changes and selects an active port for data transfers.
- **`loadbalance`**: this runner monitors traffic and uses a hash function to try to reach a perfect balance when selecting ports for packet transmission.
- **`lacp`**: implements the 802.3ad Link Aggregation Control Protocol. Can use the same transmit port selection possibilities as the **`loadbalance`** runner.

All network interaction is done through a *team* interface, composed of multiple *network port* interfaces. When controlling teamed port interfaces using NetworkManager, and especially when fault finding, keep the following in mind:

- Starting the network team interface does not automatically start the port interfaces.
- Starting a port interface always starts the teamed interface.
- Stopping the teamed interface also stops the port interfaces.
- A teamed interface without ports can start static IP connections.
- A team without ports waits for ports when starting DHCP connections.
- A team with a DHCP connection waiting for ports completes when a port with a carrier is added.
- A team with a DHCP connection waiting for ports continues waiting when a port without a carrier is added.

Configuring network teams

Figure 2.0: Configuring Network Teaming

The **nmcli** command can be used to create and manage team and port interfaces. The following four steps are used to create and activate a network team interface:

1. Create the team interface.
2. Determine the IPv4 and/or IPv6 attributes of the team interface.
3. Assign the port interfaces.
4. Bring the team and port interfaces up/down.

Create the team interface
Use the **nmcli** command to create a connection for the network team interface, with the following syntax:

```
nmcli con add type team con-name CNAME ifname INAME [config JSON]
```

where *CNAME* will be the name used to refer to the connection, *INAME* will be the interface name, and *JSON* specifies the runner to be used. *JSON* has the following syntax:

```
'{"runner": {"name": "METHOD"}}'
```

where *METHOD* is one of the following: **broadcast**, **roundrobin**, **activebackup**, **loadbalance**, or **lacp**.

Example:

```
[root@demo ~]# nmcli con add type team con-name team0 ifname team0 config '{"runner":
 {"name": "loadbalance"}}'
```

Determine the IPv4/IPv6 attributes of the team interface
Once the network team interface is created, IPv4 and/or IPv6 attributes can be assigned to it. If DHCP is available, this step is optional, because the default attributes configure the interface to get its IP settings using DHCP.

The following example demonstrates how to assign a static IPv4 address to the **team0** interface:

```
[root@demo ~]# nmcli con mod team0 ipv4.addresses 1.2.3.4/24
[root@demo ~]# nmcli con mod team0 ipv4.method manual
```

Note that the **ipv4.addresses** have to be assigned before the **ipv4.method** can be set to **manual**.

Assign the port interfaces
Use the **nmcli** command to create each of the port interfaces with the following syntax:

```
nmcli con add type team-slave con-name CNAME ifname INAME master TEAM
```

where *CNAME* will be the name used to refer to the port, *INAME* is the name of an existing interface, and *TEAM* specifies the connection name of the network team interface.

The connection name can be explicitly specified, or it will be **team-slave-*IFACE*** by default.

Example:

```
[root@demo ~]# nmcli con add type team-slave ifname eth1 master team0
[root@demo ~]# nmcli con add type team-slave ifname eth2 master team0 con-name team0-
eth2
```

Bring the team and port interfaces up/down

nmcli command can also be used to manage the connections for the team and port interfaces with the following syntax:

```
nmcli dev dis INAME
nmcli con up CNAME
```

INAME is the device name of the team or port interface to be managed. *CNAME* is the connection name of that interface. where *CNAME* is the connection name of the team or port interface to be managed.

Example:

```
[root@demo ~]# nmcli con up team0
[root@demo ~]# nmcli dev dis eth2
```

When the team interface is up, the **teamdctl** command can be used to display the team's state. The output of this command includes the status of the port interfaces.

```
[root@demo ~]# teamdctl team0 state
setup:
  runner: roundrobin
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
```

References

nmcli-examples(5), **teamdctl**(8), and **teamd**(8) man pages

Practice: Configuring Network Teaming

Guided exercise

In this lab, you will create a network team interface.

Resources:	
Machines:	**server1**

Outcomes:
A network team interface called **team0** will have a static IP address of .100/24 and will be built upon two port interfaces: **eth1** and **eth2**. It will be a fault-tolerant/active-backup interface.

Before you begin...
- Reset the server1 system.
- Become the **root** user.

1. Display the current state of the existing network interfaces. **eth1** and **eth2** will be the interfaces that will be the ports for the teamed interface.

```
[root@server1 ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 52:54:00:00:XX:0b brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 00:10:18:2b:98:85 brd ff:ff:ff:ff:ff:ff
6: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 64:31:50:18:80:8f brd ff:ff:ff:ff:ff:ff
```

2. Create a active-backup teaming interface called **team0** and assign its IPv4 settings.

   2.1. Create the **team0** connection.

```
[root@server1 ~]# nmcli con add type team con-name team0 ifname team0 config
 '{"runner": {"name": "activebackup"}}'
Connection 'team0' (5dc435ac-e4ac-403a-8e8f-163b163bf49b) successfully added.
```

   2.2. Define the IPv4 settings for **team0**. Assign it the IP address 192.168.1.100/24; the method for IP should be static.

```
[root@server1 ~]# nmcli con mod team0 ipv4.addresses '192.168.1.100/24'
[root@server1 ~]# nmcli con mod team0 ipv4.method manual
```

3. Assign **eth1** and **eth2** as port interfaces for **team0**.

```
[root@server1 ~]# nmcli con add type team-slave con-name team0-port1 ifname eth1
 master team0
```

```
Connection 'team0-port1' (f5664c4e-1dba-43f8-8427-35aee0594ed3) successfully added.
[root@server1 ~]# nmcli con add type team-slave con-name team0-port2 ifname eth2
 master team0
Connection 'team0-port2' (174e4402-b169-47d1-859f-9a4b3f30000f) successfully added.
```

4. Check the current state of the teamed interfaces on the system.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

Notice how the teamed interface immediately becomes active with **eth1** as the active port.

5. Open another terminal and **ping** the local network gateway through the **team0** interface. Let this **ping** continue to run as you perform the following steps.

```
[student@server1 ~]$ ping -I team0 192.168.1.254
PING .254 (192.168.1.254) from 192.168.1.100 team0: 56(84) bytes of data.
64 bytes from .254: icmp_seq=10 ttl=64 time=1.08 ms
64 bytes from .254: icmp_seq=11 ttl=64 time=0.789 ms
64 bytes from .254: icmp_seq=12 ttl=64 time=0.906 ms
...Output omitted...
```

6. Go back to the other **root** terminal. Bring the active port of the teamed interface down and see its impact upon **team0**.

```
[root@server1 ~]# nmcli dev dis eth1
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth2
```

The **ping** continues to work because **team0** switched over to the remaining port.

7. Bring the original port interface back up and bring the other port interface down.

```
[root@server1 ~]# nmcli con up team0-port1
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/10)
[root@server1 ~]# nmcli dev dis eth2
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

Again, note how the **ping** continues to reach the local gateway.

8. Bring the down port interface back up and observe how it affects the teamed interface, **team0**.

```
[root@server1 ~]# nmcli con up team0-port2
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/11)
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

The **ping** continues to contact the local network gateway, and the currently active port interface does not change when both port interfaces are available.

Managing Network Teaming

Objectives

After completing this section, students should be able to troubleshoot and change the configuration of a team interface.

Network teaming configuration files

NetworkManager creates configuration files for network teaming in the **/etc/sysconfig/network-scripts** the same way it does for other interfaces. A configuration file is created for each of the interfaces: for the team, and each of the ports.

The configuration file for the team interface defines the IP settings for the interface. The **DEVICETYPE** variable informs the initialization scripts this is a network team interface. Parameters for **teamd** configuration are defined in the **TEAM_CONFIG** variable. Note that the contents of **TEAM_CONFIG** uses JSON syntax.

```
# /etc/sysconfig/network-scripts/ifcfg-team0
DEVICE=team0
DEVICETYPE=Team
TEAM_CONFIG="{\"runner\": {\"name\": \"broadcast\"}}"
BOOTPROTO=none
IPADDR0=192.168.0.1
PREFIX0=24
NAME=team0
ONBOOT=yes
```

The following is an example configuration file for a team port interface.

```
# /etc/sysconfig/network-scripts/ifcfg-team0-eth1
DEVICE=eth1
DEVICETYPE=TeamPort
TEAM_MASTER=team0
NAME=team0-eth1
ONBOOT=yes
```

The **DEVICETYPE** variable informs the initialization scripts this is a team port interface. The **TEAM_MASTER** variable defines which team device it is a port for.

Setting and adjusting team configuration

Initial network team configuration is set when the team interface is created. The default runner is **roundrobin**, but a different runner can be chosen by specifying a JSON string when the team is created with the **team.config** subcommand. Default values for runner parameters are used when they are not specified.

A different runner can be assigned to an existing team, or runner parameters can be adjusted using the **nmcli con mod** command. The configuration changes can be specified as a JSON string (in the case of simple changes) or the name of a file with a more complex JSON configuration can be given.

```
nmcli con mod IFACE team.config JSON-configuration-file-or-string
```

The following example shows how to assign different priorities to port interfaces in an **active-backup** team:

```
[root@demo ~]# cat /tmp/team.conf
{
    "device": "team0",
    "mcast_rejoin": {
        "count": 1
    },
    "notify_peers": {
        "count": 1
    },
    "ports": {
        "eth1": {
     "prio": -10,
     "sticky": true,
            "link_watch": {
                "name": "ethtool"
            }
        },
        "eth2": {
     "prio": 100,
            "link_watch": {
                "name": "ethtool"
            }
        }
    },
    "runner": {
        "name": "activebackup"
    }
}
[root@demo ~]# nmcli con mod team0 team.config /tmp/team.conf
```

Note

Any changes made do not go into effect until the next time the team interface is brought up.

The **link_watch** settings in the configuration file determines how the link state of the port interfaces are monitored. The default looks like the following, and uses functionality similar to the **ethtool** command to check the link of each interface:

```
"link_watch": {
    "name": "ethtool"
}
```

Another way to check link state is to periodically use an ARP ping packet to check for remote connectivity. Local and remote IP addresses and timeouts would have to be specified. A configuration that would accomplish that would look similar to the following:

```
"link_watch":{
    "name": "arp_ping",
    "interval": 100,
    "missed_max": 30,
    "source_host": "192.168.23.2",
    "target_host": "192.168.23.1"
},
```

Note

Be aware that omitted options revert to their default values when they are not specified in the JSON file.

Troubleshooting network teams

The **teamnl** and **teamdctl** commands are very useful for troubleshooting network teams. These commands only work on network teams that are up. The following examples show some typical uses for these commands.

Display the team ports of the **team0** interface:

```
[root@demo ~]# teamnl team0 ports
 4: eth2: up 0Mbit HD
 3: eth1: up 0Mbit HD
```

Display the currently active port of **team0**:

```
[root@demo ~]# teamnl team0 getoption activeport
3
```

Set the option for the active port of **team0**:

```
[root@demo ~]# teamnl team0 setoption activeport 3
```

Use **teamdctl** to display the current state of the **team0** interface:

```
[root@demo ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

Use **teamdctl** to display the current JSON configuration for **team0**:

```
[root@demo ~]# teamdctl team0 config dump
{
    "device": "team0",
    "mcast_rejoin": {
        "count": 1
```

```
    },
    "notify_peers": {
        "count": 1
    },
    "ports": {
        "eth1": {
            "link_watch": {
                "name": "ethtool"
            },
            "prio": -10,
            "sticky": true
        },
        "eth2": {
            "link_watch": {
                "name": "ethtool"
            },
            "prio": 100
        }
    },
    "runner": {
        "name": "activebackup"
    }
}
```

References

`teamd.conf`(5) and **`teamnl`**(8) man pages

Practice: Managing Network Teaming

Guided exercise

In this lab, you will manage your network team interface. You will deactivate it, change its runner to **roundrobin**, and reactivate it. You will use **teamdctl** and **teamnl** to get information about the team interface.

Resources:	
Files:	**/etc/sysconfig/network-scripts/ifcfg-***
Machines:	**server1**

Outcomes:
A network team interface called **team0** will have a static IP address of 192.168.1.100/24 and will be built upon two port interfaces: **eth1** and **eth2**. Its runner will be changed to **roundrobin**, then changed back to **activebackup**.

Before you begin...
- Complete the Configuring Network Teaming practice exercise in the previous section. This exercise uses the teamed interfaces created in that practice exercise.
- Log into your server system and become **root**.

1. Get the initial state of the teamed interfaces on the system.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

2. Examine the configuration files created by NetworkManager that apply to the team interfaces and its ports.

   2.1. Display the file for the team interface and note how it defines the runner to be used and the IPv4 network settings for the interface.

```
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-team0
DEVICE=team0
TEAM_CONFIG="{\"runner\": {\"name\": \"activebackup\"}}"
DEVICETYPE=Team
```

```
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
NAME=team0
UUID=5dc435ac-e4ac-403a-8e8f-163b163bf49b
ONBOOT=yes
IPADDR0=192.168.1.100
PREFIX0=24
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
```

2.2. Display the configuration files for the port interfaces. Take special notice of the values of the **TEAM_MASTER** and **DEVICETYPE** shell variables.

```
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-team0-port1
BOOTPROTO=dhcp
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=team0-port1
UUID=f5664c4e-1dba-43f8-8427-35aee0594ed3
DEVICE=eth1
ONBOOT=yes
TEAM_MASTER=team0
DEVICETYPE=TeamPort
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-team0-port2
BOOTPROTO=dhcp
DEFROUTE=yes
PEERDNS=yes
PEERROUTES=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=team0-port2
UUID=174e4402-b169-47d1-859f-9a4b3f30000f
DEVICE=eth2
ONBOOT=yes
TEAM_MASTER=team0
DEVICETYPE=TeamPort
```

3. Bring the **team0** interface down and edit the configuration file to use the **roundrobin** runner.

3.1. Bring the **team0** interface down.

```
[root@server1 ~]# nmcli dev dis team0
```

3.2. Edit the configuration file for **team0** and adjust it to use the **roundrobin** runner.

```
[root@server1 ~]# vim /etc/sysconfig/network-scripts/ifcfg-team0
[root@server1 ~]# grep runner /etc/sysconfig/network-scripts/ifcfg-team0
TEAM_CONFIG="{\"runner\": {\"name\": \"roundrobin\"}}"
```

3.3. Use **nmcli** to make NetworkManager reload the updated configuration.

```
[root@server1 ~]# nmcli con load /etc/sysconfig/network-scripts/ifcfg-team0
```

4. Bring the **team0** interface back up.

4.1. Tell NetworkManager to activate **team0**.

```
[root@server1 ~]# nmcli con up team0
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/12)
```

4.2. Check the current state of **team0**.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: roundrobin
```

4.3. Log into another window and **ping** the local network gateway through the **team0** interface.

```
[root@server1 ~]# ping -I team0 192.168.1.254
PING 192.168.1.254 (192.168.1.254) from 192.168.1.100 team0: 56(84) bytes of
 data.
From 192.168.1.100 icmp_seq=1 Destination Host Unreachable
From 192.168.1.100 icmp_seq=2 Destination Host Unreachable
From 192.168.1.100 icmp_seq=3 Destination Host Unreachable
...Output omitted...
```

The **ping** command fails to contact the gateway because the teamed interface doesn't have any active ports.

5. Activate one of the port interfaces for **team0**.

5.1. Use **nmcli** to activate the **team0-port1** interface.

```
[root@server1 ~]# nmcli con up team0-port1
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/13)
```

5.2. Display the state of the teamed interface.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: roundrobin
ports:
```

```
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
```

5.3. **ping** the local network gateway through the **team0** interface. It should be able to reach the gateway.

```
[root@server1 ~]# ping -I team0 192.168.1.254
PING 192.168.1.254 (192.168.1.254) from 192.168.1.100 team0: 56(84) bytes of
 data.
64 bytes from 192.168.1.254: icmp_seq=1 ttl=64 time=0.516 ms
64 bytes from 192.168.1.254: icmp_seq=2 ttl=64 time=0.703 ms
64 bytes from 192.168.1.254: icmp_seq=3 ttl=64 time=0.422 ms
```

6. Bring up the other team port for **team0**.

6.1. Use **nmcli** to activate the **team0-port2** interface.

```
[root@server1 ~]# nmcli con up team0-port2
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/14)
```

6.2. Display the updated state of the teamed interface.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: roundrobin
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
```

7. Use **teamdctl** to display the configuration for **team0**.

```
[root@server1 ~]# teamdctl team0 config dump
{
    "device": "team0",
    "ports": {
        "eth1": {
            "link_watch": {
                "name": "ethtool"
            }
        },
        "eth2": {
            "link_watch": {
```

```
                "name": "ethtool"
            }
        }
    },
    "runner": {
        "name": "roundrobin"
    }
}
```

8. Use the **teamnl** command to display the tunable options for **team0**.

```
[root@server1 ~]# teamnl team0 options
 queue_id (port:eth2) 0
 priority (port:eth2) 0
 user_linkup_enabled (port:eth2) false
 user_linkup (port:eth2) true
 enabled (port:eth2) true
 queue_id (port:eth1) 0
 priority (port:eth1) 0
 user_linkup_enabled (port:eth1) false
 user_linkup (port:eth1) true
 enabled (port:eth1) true
 mcast_rejoin_interval 0
 mcast_rejoin_count 0
 notify_peers_interval 0
 notify_peers_count 0
 mode roundrobin
```

9. Modify the **team0** interface so it uses the **activebackup** runner instead of **roundrobin**.

   9.1. The interface can only be modified after it is brought down.

```
[root@server1 ~]# nmcli dev dis team0
```

   9.2. Use **nmcli** to tune the teamed interface to use the **activebackup** runner.

```
[root@server1 ~]# nmcli con mod team0 team.config '{"runner": {"name":
 "activebackup"}}'
```

   9.3. Examine the changes made to the interface's configuration file.

```
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-team0
DEVICE=team0
TEAM_CONFIG="{\"runner\": {\"name\": \"activebackup\"}}"
DEVICETYPE=Team
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
NAME=team0
UUID=5dc435ac-e4ac-403a-8e8f-163b163bf49b
ONBOOT=yes
IPADDR0=192.168.1.100
PREFIX0=24
```

```
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
```

10. Reactivate the teamed interface and both of its port interfaces.

    10.1 Use **nmcli** to activate the teamed interface.

```
[root@server1 ~]# nmcli con up team0
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/15)
```

    10.2 Display its initial state. Note that the port interfaces didn't get activated.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
runner:
  active port:
```

    10.3 Activate the two port interfaces and display the resulting teamed interface state.

```
[root@server1 ~]# nmcli con up team0-port1
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/16)
[root@server1 ~]# nmcli con up team0-port2
Connection successfully activated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/17)
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

    Restart **ping** and make sure it works once the first port interface is brought back up.

```
[root@server1 ~]# ping -I team0 192.168.1.254
PING 192.168.1.254 (192.168.1.254) from 192.168.1.100 team0: 56(84) bytes of
 data.
64 bytes from 192.168.1.254: icmp_seq=1 ttl=64 time=0.516 ms
64 bytes from 192.168.1.254: icmp_seq=2 ttl=64 time=0.703 ms
64 bytes from 192.168.1.254: icmp_seq=3 ttl=64 time=0.422 ms
```

11. Use the **teamnl** to display the options available to an **activebackup** team device.

```
[root@server1 ~]# teamnl team0 options
 queue_id (port:eth2) 0
 priority (port:eth2) 0
 user_linkup_enabled (port:eth2) false
 user_linkup (port:eth2) true
 enabled (port:eth2) false
 queue_id (port:eth1) 0
 priority (port:eth1) 0
 user_linkup_enabled (port:eth1) false
 user_linkup (port:eth1) true
 enabled (port:eth1) true
 activeport 3
 mcast_rejoin_interval 0
 mcast_rejoin_count 1
 notify_peers_interval 0
 notify_peers_count 1
 mode activebackup
```

Configuring Software Bridges

Objectives

After completing this section, students should be able to configure and troubleshoot local software bridges and associated interfaces.

Software bridges

A network bridge is a link-layer device that forwards traffic between networks based on MAC addresses. It learns what hosts are connected to each network, builds a table of MAC addresses, then makes packet forwarding decisions based on that table. A software bridge can be used in a Linux environment to emulate a hardware bridge. The most common application for software bridges is in virtualization applications for sharing a hardware NIC among one or more virtual NICs.

Configure software bridges

The **nmcli** can be used to configure software bridges persistently. First, the software bridge is created, then existing interfaces are connected to it. For example, the following commands will create a bridge called **br0** and attach both the **eth1** and **eth2** interfaces to it.

```
[root@demo ~]# nmcli con add type bridge con-name br0 ifname br0
[root@demo ~]# nmcli con add type bridge-slave con-name br0-port1 ifname eth1 master br0
[root@demo ~]# nmcli con add type bridge-slave con-name br0-port2 ifname eth2 master br0
```

Note

NetworkManager can only attach Ethernet interfaces to a bridge. It does not support aggregate interfaces, such as a teamed or bonded interface. These must be configured by manipulating the configuration files in **/etc/sysconfig/network-scripts**.

Software bridge configuration files

Software bridges are managed by interface configuration files found in the **/etc/sysconfig/network-scripts** directory. There is an **ifcfg-*** configuration file for each software bridge.

The following is a sample configuration file for a software bridge:

```
# /etc/sysconfig/network-scripts/ifcfg-br1
DEVICE=br1
NAME=br1
TYPE=Bridge
BOOTPROTO=none
IPADDR0=192.168.0.1
PREFIX0=24
STP=yes
BRIDGING_OPTS=priority=32768
```

The **TYPE=Bridge** definition specifies that this is a software bridge. **BRIDGING_OPTS** defines additional bridge options. Note that this bridge has been assigned a static IP address.

The following configuration file attaches an Ethernet interface to a software bridge:

```
# /etc/sysconfig/network-scripts/ifcfg-br1-port0
TYPE=Ethernet
NAME=br1-port0
DEVICE=eth1
ONBOOT=yes
BRIDGE=br1
```

The single variable definition, **BRIDGE=br1**, is what ties this interface to the software bridge, **br1**.

Note

To implement a software bridge on an existing teamed or bonded network interface managed by NetworkManager, NetworkManager will have to be disabled since it only supports bridges on simple Ethernet interfaces. Configuration files for the bridge will have to be created by hand. The **ifup** and **ifdown** utilities can be used to manage the software bridge and other network interfaces.

The **brctl show** command will display software bridges and the list of interfaces attached to them.

```
[root@demo ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
br1             8000.52540001050b       yes             eth1
```

References

nmcli-examples(5) and **brctl**(8) man pages

Practice: Configuring Software Bridges

Guided exercise

In this lab, you will create a network bridge.

Resources:	
Files:	**/etc/sysconfig/network-scripts/ifcfg-***
Machines:	**server1**

Outcomes:
A network bridge called **br1** will be attached to the **eth1** network interface. It will have a static IP address of 192.168.1.100/24.

Before you begin...
- Reset the server1 system.
- Become the **root** user.

1. Define a software bridge called **br1** and assign it a static IPv4 address of 192.168.1.100/24.

   1.1. Use **nmcli** to create the software bridge.

```
[root@server1 ~]# nmcli con add type bridge con-name br1 ifname br1
Connection 'br1' (d9d56520-574a-4e2a-9f43-b593a1bdff61) successfully added.
```

   1.2. Configure the IPv4 addressing for the interface.

```
[root@server1 ~]# nmcli con mod br1 ipv4.addresses 192.168.1.100/24
[root@server1 ~]# nmcli con mod br1 ipv4.method manual
```

2. Attach the **eth1** interface to the **br1** software bridge.

```
[root@server1 ~]# nmcli con add type bridge-slave con-name br1-port0 ifname eth1
 master br1
Connection 'br1-port0' (5f5e7ea8-b507-4c10-a61f-779369cf82ee) successfully added.
```

3. Inspect the configuration files that were created for the software bridge by NetworkManager. Look for variables that connect the two interfaces.

```
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-br1
DEVICE=br1
STP=yes
TYPE=Bridge
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
```

```
NAME=br1
UUID=d9d56520-574a-4e2a-9f43-b593a1bdff61
ONBOOT=yes
IPADDR0=192.168.1.100
PREFIX0=24
BRIDGING_OPTS=priority=32768
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
[root@server1 ~]# cat /etc/sysconfig/network-scripts/ifcfg-br1-port0
TYPE=Ethernet
NAME=br1-port0
UUID=5f5e7ea8-b507-4c10-a61f-779369cf82ee
DEVICE=eth1
ONBOOT=yes
BRIDGE=br1
```

4. Display the link status of the network interfaces.

```
[root@server1 ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 52:54:00:00:XX:0b brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br1
 state UP mode DEFAULT qlen 1000
    link/ether 00:10:18:2b:98:85 brd ff:ff:ff:ff:ff:ff
6: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 64:31:50:18:80:8f brd ff:ff:ff:ff:ff:ff
7: br1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode
 DEFAULT
    link/ether 00:10:18:2b:98:85 brd ff:ff:ff:ff:ff:ff
```

5. Use **`brctl`** to display information about software bridges on the system.

```
[root@server1 ~]# brctl show
bridge name     bridge id               STP enabled     interfaces
br1             8000.52540001050b       yes             eth1
```

6. Ping the local network gateway using the software bridge.

```
[root@server1 ~]# ping -I br1 192.168.1.254
PING 192.168.1.254 (192.168.1.254) from 192.168.1.100 br1: 56(84) bytes of data.
64 bytes from 192.168.1.254: icmp_seq=10 ttl=64 time=0.520 ms
64 bytes from 192.168.1.254: icmp_seq=11 ttl=64 time=0.470 ms
64 bytes from 192.168.1.254: icmp_seq=12 ttl=64 time=0.339 ms
64 bytes from 192.168.1.254: icmp_seq=13 ttl=64 time=0.294 ms
...Output omitted...
```

Lab: Configuring Link Aggregation and Bridging

Performance checklist

In this lab, you will create a bridge that is connected to a network team interface.

Resources:	
Files:	**/etc/sysconfig/network-scripts/ifcfg-***
Machines:	**server1**

Outcomes:
server1 will have an **activebackup** team interface, called **team0**. The **team0** interface will be built upon the port interfaces **eth1** and **eth2**. **team0** will be attached to a bridge, called **brteam0**. The bridge will have a static IP address of 192.168.1.100/24.

Before you begin...

- Reset the server1 system.
- Become the **root** user.

1. Confirm that **eth1** and **eth2** are available for use. Display the current state of the existing network interfaces. **eth1** and **eth2** will be the port interfaces that will be teamed into a single interface.
2. Create an **activebackup** network team interface called **team0**.
3. Disable the **team0** device then the NetworkManager service.
4. Manipulate the interface configuration files so that the **team0** interface is attached to a software bridge called **brteam0**. The bridge should have a static IP address of 192.168.1.100/24.
5. Reset the network to start the new bridge, **brteam0**, and reactivate the **team0** interface.

Note

You may see an error **Job for network.service failed. See 'systemctl status network.service' and 'journalctl -xn' for details.**. This should not happen if you do a clean reboot. You can ignore this for now.

6. Test the network configuration.

Solution

In this lab, you will create a bridge that is connected to a network team interface.

Resources:	
Files:	**/etc/sysconfig/network-scripts/ifcfg-***
Machines:	**server1**

Outcomes:
server1 will have an **activebackup** team interface, called **team0**. The **team0** interface will be built upon the port interfaces **eth1** and **eth2**. **team0** will be attached to a bridge, called **brteam0**. The bridge will have a static IP address of 192.168.1.100/24.

Before you begin...
- Reset the server1 system.
- Become the **root** user.

1. Confirm that **eth1** and **eth2** are available for use. Display the current state of the existing network interfaces. **eth1** and **eth2** will be the port interfaces that will be teamed into a single interface.

```
[root@server1 ~]# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 52:54:00:00:XX:0b brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 00:10:18:2b:98:85 brd ff:ff:ff:ff:ff:ff
6: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 DEFAULT qlen 1000
    link/ether 64:31:50:18:80:8f brd ff:ff:ff:ff:ff:ff
```

2. Create an **activebackup** network team interface called **team0**.

   2.1. Create the network team interface and call it **team0**.

```
[root@server1 ~]# nmcli con add type team con-name team0 ifname team0 config
 '{"runner": {"name": "activebackup"}}'
Connection 'team0' (2f608473-ff8b-4a0d-b250-79567e3f4a13) successfully added.
```

   2.2. Assign **eth1** and **eth2** as network port interfaces for **team0**.

```
[root@server1 ~]# nmcli con add type team-slave con-name team0-port1 ifname eth1
 master team0
Connection 'team0-port1' (3367d0ef-deb5-444b-bc01-0ed3825615a9) successfully
 added.
[root@server1 ~]# nmcli con add type team-slave con-name team0-port2 ifname eth2
 master team0
Connection 'team0-port2' (4951d25b-a454-4735-8c7b-a51b983df56b) successfully
 added.
```

2.3. Confirm the **team0** interface is up and working properly.

```
[root@server1 ~]# teamdctl team0 state
setup:
  runner: activebackup
ports:
  eth1
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
  eth2
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
runner:
  active port: eth1
```

3. Disable the **team0** device then the NetworkManager service.

```
[root@server1 ~]# nmcli dev dis team0
[root@server1 ~]# systemctl stop NetworkManager
[root@server1 ~]# systemctl disable NetworkManager
rm '/etc/systemd/system/multi-user.target.wants/NetworkManager.service'
rm '/etc/systemd/system/dbus-org.freedesktop.NetworkManager.service'
rm '/etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service'
```

4. Manipulate the interface configuration files so that the **team0** interface is attached to a software bridge called **brteam0**. The bridge should have a static IP address of 192.168.1.100/24.

4.1. List the original interface configuration files that were created by NetworkManager:

```
[root@server1 ~]# cd /etc/sysconfig/network-scripts/
[root@server1 network-scripts]# ls -1 ifcfg-*
ifcfg-eth0
ifcfg-lo
ifcfg-team0
ifcfg-team0-port1
ifcfg-team0-port2
```

4.2. Edit **ifcfg-team0** to define a **BRIDGE** variable that assigns it to the new bridge about to be created, **brteam0**.

```
[root@server1 network-scripts]# echo "BRIDGE=brteam0" >> ifcfg-team0
[root@server1 network-scripts]# tail ifcfg-team0
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
IPV6_FAILURE_FATAL=no
NAME=team0
```

```
UUID=2f608473-ff8b-4a0d-b250-79567e3f4a13
ONBOOT=yes
BRIDGE=brteam0
```

4.3. Remove the IP configuration information from the configuration files for the team port interfaces. Replace all text in **ifcfg-team0-port1** with the below text.

```
[root@server1 network-scripts]# vim ifcfg-team0-port1
[root@server1 network-scripts]# cat ifcfg-team0-port1
NAME=team0-port1
DEVICE=eth1
ONBOOT=yes
TEAM_MASTER=team0
DEVICETYPE=TeamPort
```

4.4. Remove the IP configuration information from the configuration files for the team port interfaces. Replace all text in **ifcfg-team0-port2** with the below text.

```
[root@server1 network-scripts]# vim ifcfg-team0-port2
[root@server1 network-scripts]# cat ifcfg-team0-port2
NAME=team0-port2
DEVICE=eth2
ONBOOT=yes
TEAM_MASTER=team0
DEVICETYPE=TeamPort
```

4.5. Create a new interface configuration file for the bridge, **ifcfg-brteam0**. Define the IP configuration information in that file. Replace the contents of **ifcfg-brteam0** with the text below.

```
[root@server1 network-scripts]# vim ifcfg-brteam0
[root@server1 network-scripts]# cat ifcfg-brteam0
DEVICE=brteam0
ONBOOT=yes
TYPE=Bridge
IPADDR0=192.168.1.100
PREFIX0=24
```

5. Reset the network to start the new bridge, **brteam0**, and reactivate the **team0** interface.

```
[root@server1 network-scripts]# systemctl restart network
```

Note

You may see an error **Job for network.service failed. See 'systemctl status network.service' and 'journalctl -xn' for details.**. This should not happen if you do a clean reboot. You can ignore this for now.

6. Test the network configuration.

- Use **ping** to see if the local gateway, 192.168.1.254, can be reached through the **brteam0** interface.

```
[root@server1 ~]# ping -I brteam0 192.168.1.254
PING 192.168.1.254 (192.168.1.254) from 192.168.1.100 brteam0: 56(84) bytes of
 data.
64 bytes from 192.168.1.254: icmp_seq=1 ttl=64 time=0.172 ms
64 bytes from 192.168.1.254: icmp_seq=2 ttl=64 time=0.091 ms
64 bytes from 192.168.1.254: icmp_seq=3 ttl=64 time=0.052 ms
... Output omitted ...
```

Summary

Configuring Network Teaming

In this section, students learned how to define network teaming interfaces.

Managing Network Teaming

In this section, students learned how to:

- Navigate and modify network teaming configuration files.
- Modify team runner configuration settings.
- Use **`teamnl`** and **`teamdctl`** to troubleshoot network teaming.

Configuring Software Bridges

In this section, students learned how to define software bridge network interfaces.

CHAPTER 3

NETWORK PORT SECURITY

Overview	
Goal	To permit and reject access to network services using advanced SELinux and firewalld filtering techniques.
Objectives	• Review firewalld concepts and management commands covered in previous courses. • Configure more complex firewall configurations using firewalld's support for "rich language rules." • Describe and implement Network Address Translation (NAT). • Ensure network ports have the correct SELinux type so that services are able to bind to them.
Sections	• Managing Firewalld (and Practice) • Managing Rich Rules (and Practice) • Masquerading and Port Forwarding (and Practice) • Managing SELinux Port Labeling (and Practice)
Lab	• Network Port Security

Managing Rich Rules

Objectives

After completing this section, students should be able to configure more complex firewall configurations using firewalld's support for "rich language rules."

Rich rules concepts

Apart from the regular zones and services syntax that **firewalld** offers, administrators have two other options for adding firewall rules: *direct rules* and *rich rules*.

Direct rules

Direct rules allow an administrator to insert hand-coded **{ip,ip6,eb}tables** rules into the zones managed by **firewalld**. While powerful, and exposing features of the kernel **netfilter** subsystem not exposed through other means, these rules can be hard to manage. Direct rules also offer less flexibility than standard rules and rich rules. Configuring direct rules is not covered in this course, but documentation is available in the **firewall-cmd**(1) and **firewalld.direct**(5) man pages for those administrators who are already familiar with **{ip,ip6,eb}tables** syntax.

Unless explicitly inserted into a zone managed by **firewalld**, direct rules will be parsed before any **firewalld** rules are.

A short example of adding some direct rules to blacklist an IP range:

```
[root@server1 ~]# firewall-cmd --direct --permanent --add-chain ipv4 raw blacklist
[root@server1 ~]# firewall-cmd --direct --permanent --add-rule ipv4 raw PREROUTING 0 -s
 192.168.0.0/24 -j blacklist
[root@server1 ~]# firewall-cmd --direct --permanent --add-rule ipv4 raw blacklist 0 -m
 limit --limit 1/min -j LOG --log-prefix "blacklisted "
[root@server1 ~]# firewall-cmd --direct --permanent --add-rule ipv4 raw blacklist 1 -j
 DROP
```

Rich rules

firewalld *rich rules* give administrators an expressive language in which to express custom firewall rules that are not covered by the basic **firewalld** syntax; for example, to only allow connections to a service from a single IP address, instead of all IP addresses routed through a zone.

Rich rules can be used to express basic allow/deny rules, but can also be used to configure logging, both to **syslog** and **auditd**, as well as port forwards, masquerading, and rate limiting.

The basic syntax of a rich rule can be expressed by the following block:

```
rule
  [source]
  [destination]
  service|port|protocol|icmp-block|masquerade|forward-port
  [log]
  [audit]
  [accept|reject|drop]
```

Almost every single element of a rule can take additional arguments in the form of **option=value**.

Note

For the full available syntax for rich rules, consult the **firewalld.richlanguage**(5) man page.

Rule ordering
Once multiple rules have been added to a zone (or the firewall in general), the ordering of rules can have a big effect on how the firewall behaves.

The basic ordering of rules inside a zone is the same for all zones:

1. Any port forwarding and masquerading rules set for that zone.
2. Any logging rules set for that zone.
3. Any deny rules set for that zone. Any allow rules set for that zone.
4. Any allow rules set for that zone.

In all cases, the first match will win. If a packet has not been matched by any rule in a zone, it will typically be denied, but zones might have a different default; for example, the **trusted** zone will **accept** any unmatched packet. Also, after matching a logging rule, a packet will continue to be processed as normal.

Direct rules are an exception. Most direct rules will be parsed before any other processing is done by **firewalld**, but the direct rule syntax allows an administrator to insert any rule they want anywhere in any zone.

Testing and debugging
To make testing and debugging easier, almost all rules can be added to the *runtime* configuration with a timeout. The moment the rule with a timeout is added to the firewall, the timer starts counting down for that rule. Once the timer for a rule has reached zero seconds, that rule is removed from the *runtime* configuration.

Using timeouts can be an incredibly useful tool while working on a remote firewalls, especially when testing more complicated rule sets. If a rule works, the administrator can add it again, but with the **--permanent** option (or at least without a timeout). If the rule does not work as intended, maybe even locking the administrator out of the system, it will be removed automatically, allowing the administrator to continue his or her work.

A timeout is added to a runtime rule by adding the option **--timeout=<*TIMEINSECONDS*>** to the end of the **firewall-cmd** that enables the rule.

Working with rich rules

firewall-cmd has four options for working with rich rules. All of these options can be used in combination with the regular **--permanent** or **--zone=<*ZONE*>** options.

Option	Explanation
--add-rich-rule='<*RULE*>'	Add **<*RULE*>** to the specified zone, or the default zone if no zone is specified.

Option	Explanation
--remove-rich-rule='<*RULE*>'	Remove **<*RULE*>** to the specified zone, or the default zone if no zone is specified.
--query-rich-rule='<*RULE*>'	Query if **<*RULE*>** has been added to the specified zone, or the default zone if no zone is specified. Returns **0** if the rule is present, otherwise **1**.
--list-rich-rules	Outputs all rich rules for the specified zone, or the default zone if no zone is specified.

Any configured rich rules are also shown in the output from **firewall-cmd --list-all** and **firewall-cmd --list-all-zones**.

Rich rules examples
Some examples of rich rules:

- ```
  [root@server1 ~]# firewall-cmd --permanent --zone=classroom --add-rich-rule='rule
   family=ipv4 source address=192.168.0.11/32 reject'
  ```

  Reject all traffic from the IP address **192.168.0.11** in the **classroom** zone.

  When using **source** or **destination** with an **address** option, the **family=** option of **rule** must be set to either **ipv4** or **ipv6**.

- ```
  [root@server1 ~]# firewall-cmd --add-rich-rule='rule service name=ftp limit value=2/m
   accept'
  ```

  Allow two new connections to **ftp** per minute in the default zone.

  Note that this change is only made in the *runtime* configuration.

- ```
  [root@server1 ~]# firewall-cmd --permanent --add-rich-rule='rule protocol value=esp
   drop'
  ```

  Drop all incoming IPsec **esp** protocol packets from anywhere in the default zone.

Note

The difference between **reject** and **drop** lies in the fact that a **reject** will send back an ICMP packet detailing that, and why, a connection was rejected. A **drop** just drops the packet and does nothing else. Normally an administrator will want to use **reject** for friendly and neutral networks, and **drop** only for hostile networks.

- ```
  [root@server1 ~]# firewall-cmd --permanent --zone=vnc --add-rich-rule='rule
   family=ipv4 source address=192.168.1.0/24 port port=7900-7905 protocol=tcp accept'
  ```

  Accept all TCP packets on ports **7900**, up to and including port **7905**, in the **vnc** zone for the **192.168.1.0/24** subnet.

Logging with rich rules

When debugging, or monitoring, a firewall, it can be useful to have a log of accepted or rejected connections. **firewalld** can accomplish this in two ways: by logging to **syslog**, or by sending messages to the kernel **audit** subsystem, managed by **auditd**.

In both cases, logging can be *rate limited*. Rate limiting ensures that system log files do not fill up with messages at a rate such that the system cannot keep up, or fills all its disk space.

The basic syntax for logging to **syslog** using rich rules is:

```
log [prefix="<PREFIX TEXT>" [level=<LOGLEVEL>] [limit value="<RATE/DURATION>"]
```

Where **<LOGLEVEL>** is one of **emerg**, **alert**, **crit**, **error**, **warning**, **notice**, **info**, or **debug**.

<DURATION> can be one of **s** for seconds, **m** for minutes, **h** for hours, or **d** for days. For example, **limit value=3/m** will limit the log messages to a maximum of three per minute.

The basic syntax for logging to the audit subsystem is:

```
audit [limit value="<RATE/DURATION>"]
```

Rate limiting is configured in the same way as for **syslog** logging.

Logging examples

Some examples of logging using rich rules:

- ```
  [root@server1 ~]# firewall-cmd --permanent --zone=work --add-rich-rule='rule service
   name="ssh" log prefix="ssh " level="notice" limit value="3/m" accept
  ```

  Accept new connections to **ssh** from the **work** zone, log new connections to **syslog** at the **notice** level, and with a maximum of three message per minute.

- ```
  [root@server1 ~]# firewall-cmd --add-rich-rule='rule family=ipv6 source
   address="2001:db8::/64" service name="dns" audit limit value="1/h" reject' --
  timeout=300
  ```

  New IPv6 connections from the subnet **2001:db8::/64** in the default zone to DNS are rejected for the next five minutes, and rejected connections are logged to the **audit** system with a maximum of one message per hour.

References

firewalld.richlanguage(5), **firewall-cmd**(1), and **firewalld.direct**(5) man pages

Practice: Writing Custom Rules

Guided exercise

In this lab, you will configure your **server1** system to allow connections to a (new) **http** service, but only from **desktop1**, and with a rate-limited log message.

Resources:	
Machines:	**desktop1** and **server1**

Outcomes:
Custom firewall rules that configure rate-limited logging for specific connections.

Before you begin...
- Reset your **server1** system.

Your company is running a trial that includes starting a web server on **server1**, but for the duration of the trial, only **desktop1** should be able to connect. Since this could potentially generate many log entries, this logging should be limited to a maximum of three messages per second, and all log messages should be prefixed with the message "**NEW HTTP** ".

It has been decided that you, the IT Rock Star, will implement this using firewalld *rich rules*.

1. First install, start, and enable **httpd**.

   1.1.
   ```
   [root@server1 ~]# yum -y install httpd
   ```

   1.2.
   ```
   [root@server1 ~]# systemctl start httpd.service
   ```

   1.3.
   ```
   [root@server1 ~]# systemctl enable httpd.service
   ```

2. Configure a firewall rule in the default zone that allows traffic to **http** only from your **desktop1** system. This traffic should be logged, but with a maximum of three new connections per second.

   2.1. Permanently create the new firewall rule.

   ```
   [root@server1 ~]# firewall-cmd --permanent --add-rich-rule='rule family=ipv4
    source address=192.168.0.1/32 service name="http" log level=notice prefix="NEW
    HTTP " limit value="3/s" accept'
   ```

   2.2. Activate the changes to your firewall.

   ```
   [root@server1 ~]# firewall-cmd --reload
   ```

   2.3. On your **server1** system, use **tail -f** to view the additions to **/var/log/messages** in real time.

```
[root@server1 ~]# tail -f /var/log/messages
```

2.4. From your **desktop1** system, use **curl** to connect to the **httpd** service running on **server1**.

```
[student@desktop1 ~]$ curl http://server1.example.com
```

2.5. Inspect the output of your running **tail** command on **server1**. You should see a message for the new connection like this:

```
May  9 08:04:11 server1 kernel: NEW HTTP IN=eth0 OUT= MAC=... SRC=192.168.0.1
 DST=192.168.0.101 LEN=60....
```

Masquerading and Port Forwarding

Objectives

After completing this section, students should be able to describe and implement Network Address Translation (NAT).

Network Address Translation (NAT)

`firewalld` supports two types of *Network Address Translation* (NAT): *masquerading* and *port forwarding*. Both can be configured on a basic level with regular **`firewall-cmd`** rules, and more advanced forwarding configurations can be accomplished with rich rules. Both forms of NAT modify certain aspects of a packet, like the source or destination, before sending it on.

Masquerading

With *masquerading*, a system will forward packets that are not directly addressed to itself to the intended recipient, while changing the *source address* of the packets that go through to its own public IP address. When answers to those packets come in, the firewall will then modify the destination address to the address of the original host, and send the packet on. This is usually used on the edge of a network to provide Internet access to an internal network. Masquerading is a form of *Network Address Translation* (NAT).

Important

Masquerading can only be used with **`IPv4`**, not with **`IPv6`**.

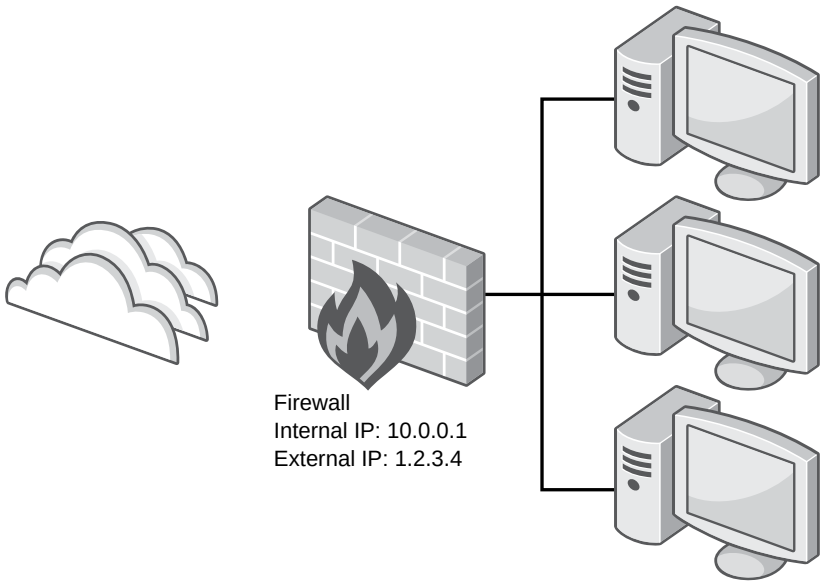

Figure 3.1: A sample network layout for NAT

An example of how masquerading works based on the network layout described in *Figure 3.1: A sample network layout for NAT*.

1. One of the machines behind the firewall sends a packet to an address outside of the local network. The packet has a source address of **10.0.0.100** (the address of the machine), and a destination address of **2.17.39.214**.

2. Since the destination address is not on the local subnet, the packet will be routed to the default gateway configured on the source machine; in this case, **10.0.0.1**, the IP address of the firewall.

3. The firewall accepts the packet, changes the source address to **1.2.3.4** (the external IP for the firewall), stores a reference to this connection in its connection state table, then passes it to a router on the Internet based on its routing table.

4. An answer to the packet comes back from the Internet. The router looks up the connection in its connection state table, then changes the destination address to **10.0.0.100** (the original sender), and passes the packet on.

5. The original sender receives the answer to its request.

Configuring masquerading

To configure masquerading for a zone with regular **firewall-cmd** commands, use the following syntax:

```
[root@server1 ~]# firewall-cmd --permanent --zone=<ZONE> --add-masquerade
```

This will masquerade any packets sent to the firewall from clients defined in the sources for that zone (both interfaces and subnets) that are not addressed to the firewall itself.

To gain more control over what clients will be masqueraded, a rich rule can be used as well.

```
[root@server1 ~]# firewall-cmd --permanent --zone=<ZONE> --add-rich-rule='rule
 family=ipv4 source address=192.168.0.0/24 masquerade'
```

Port forwarding

Another form of NAT is *port forwarding*. With port forwarding, traffic to a single port is forwarded either to a different port on the same machine, or to a port on a different machine. This mechanism is typically used to “hide” a server behind another machine, or to provide access to a service on an alternate port.

Important

When a port forward is configured to forward packets to a different machine, any replies from that machine will normally be sent directly to the original client from that machine. Since this will result in an invalid connection on most configurations, the machine that is forwarded to will have to be masqueraded through the firewall that performed the port forwarding.

A common configuration is to forward a port from the firewall machine to a machine that is already masqueraded behind the firewall.

An example of a port forward based on the network layout described in *Figure 3.1: A sample network layout for NAT*. Assume that the machine with the IP address **10.0.0.100** behind the

firewall is running a web server on port **8080/TCP**, and that the firewall is configured to forward traffic coming in on port **80/TCP** on its external interface to port **8080/TCP** on that machine.

1. A client from the Internet sends a packet to port **80/TCP** on the external interface of the firewall.
2. The firewall changes the destination address and port of this packet to **10.0.0.100** and **8080/TCP** and forwards it on. The source address and port remain unchanged.
3. The machine behind the firewall sends a response to this packet. Since this machine is being masqueraded (and the firewall is configured as the default gateway), this packet is sent to the original client, appearing to come from the external interface on the firewall.

Configuring port forwarding

To configure port forwarding with regular **firewall-cmd** commands, use the following syntax:

```
[root@server1 ~]# firewall-cmd --permanent --zone=<ZONE> --add-forward-
port=port=<PORTNUMBER>:proto=<PROTOCOL>[:toport=<PORTNUMBER>][:toaddr=<IPADDR>]
```

Both the **toport=** and **toaddr=** parts are optional, but *at least* one of those two will need to be specified.

As an example, the following command will forward incoming connections on port **513/TCP** on the firewall to port **132/TCP** on the machine with the IP address **192.168.0.254** for clients from the **public** zone:

```
[root@server1 ~]# firewall-cmd --permanent --zone=public --add-forward-
port=port=513:proto=tcp:toport=132:toaddr=192.168.0.254
```

To gain more control over port forwarding rules, the following syntax can be used with rich rules:

```
forward-port port=<PORTNUM> protocol=tcp|udp [to-port=<PORTNUM>] [to-addr=<ADDRESS>]
```

An example that uses rich rules to forward traffic from **192.168.0.0/26** in the **work** zone to port **80/TCP** to port **8080/TCP** on the firewall machine itself:

```
[root@server1 ~]# firewall-cmd --permanent --zone=work --add-rich-rule='rule family=ipv4
 source address=192.168.0.0/26 forward-port port=80 protocol=tcp to-port=8080'
```

References

firewalld.richlanguage(5) and **firewall-cmd**(1) man pages

Practice: Forwarding a Port

Guided exercise

In this lab, you will configure your **server1** system to forward a request to port **443/tcp** from **desktop1** to port **22/tcp**.

Resources:	
Machines:	**desktop1** and **server1**

Outcomes:
Custom firewall rules that configure a port forward.

Before you begin...
- Reset your **server1** system.

Your company is running a trial for a new bastion host. As part of this trial, your **desktop1** should be able to connect to the SSH daemon on your **server1** system on port **443/tcp**. Since this is purely a trial, you do not wish to bind **sshd** to that port directly, and only your **desktop1** should be able to connect using port **443/tcp**.

It has been decided that you, the chosen one, will implement this using firewalld *rich rules*.

- Configure the firewall on **server1** to forward port **443/tcp** to **22/tcp**, but only for your **desktop1** machine. The IP address of your **desktop1** machine is **192.168.0.1**.

  1.1. Permanently add the port forwarding firewall rule on **server1**.

```
[root@server1 ~]# firewall-cmd --permanent --add-rich-rule 'rule family=ipv4
 source address=192.168.0.1/32 forward-port port=443 protocol=tcp to-port=22'
```

  1.2. Reload the firewall configuration to activate your changes.

```
[root@server1 ~]# firewall-cmd --reload
```

  1.3. Test if **sshd** is now available on port **443/tcp** from your **desktop1** system.

```
[student@desktop1 ~]$ ssh -p 443 server1.example.com
The authenticity of host '[server1.example.com]:443 ([192.168.0.101]:443)' can't
 be established.
ECDSA key fingerprint is XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX.
Are you sure you want to continue connecting (yes/no)? yes
student@server1.example.com's password: student
```

Managing SELinux Port Labeling

Objectives

After completing this section, students should be able to ensure network ports have the correct SELinux type so that services are able to bind to them.

SELinux port labeling

SELinux does more than just file and process labeling. Network traffic is also tightly enforced by the SELinux policy. One of the methods that SELinux uses for controlling network traffic is labeling network ports; for example, in the **targeted** policy, port **22/TCP** has the label **ssh_port_t** associated with it.

Whenever a process wants to listen on a port, SELinux will check to see if the label associated with that process (the domain) is allowed to bind that port label. This can stop a rogue service from taking over ports otherwise used by other (legitimate) network services.

Managing SELinux port labeling

Whenever an administrator decides to run a service on a nonstandard port, there is a high chance that SELinux port labels will need to be updated. In some cases, the **targeted** policy has already labeled the port with a type that can be used; for example, since port **8008/TCP** is often used for web applications, that port is already labeled with **http_port_t**, the default port type for the web server.

Listing port labels

To get an overview of all the current port label assignments, administrators can use the **port** subcommand of the **semanage** command. The **-l** option will list all current assignments, in the form:

```
port_label_t tcp|udp comma,separated,list,of,ports
```

To only view local changes to the default policy, administrators can add the **-C** option to this command.

Example output:

```
[root@server1 ~]# semanage port -l
...
http_cache_port_t       tcp   8080, 8118, 8123, 10001-10010
http_cache_port_t       udp   3130
http_port_t             tcp   80, 81, 443, 488, 8008, 8009, 8443, 9000
...
```

Note that a port label can appear twice in the output, once for **TCP** and once for **UDP**.

Note

A graphical tool, **system-config-selinux**, is also available for administrators who prefer to work with GUI tools. This tool is part of the *policycoreutils-gui* package.

Managing port labels

semanage can also be used to assign new port labels, remove port labels, or modify existing ones.

Important

Only local modifications can be removed or modified. To allow a service to bind to a port label normally associated with another service, a small policy module must be written. Writing and generating policy modules falls outside the scope of this course. It is also not possible to remove a port label from the policy without overriding the policy module that provided that label.

To add a port to an existing port label (type), use the following syntax:

```
[root@server1 ~]# semanage port -a -t port_label -p tcp|udp PORTNUMBER
```

For example, to allow a **gopher** service to listen on port **71/TCP**:

```
[root@server1 ~]# semanage port -a -t gopher_port_t -p tcp 71
```

Note

The **targeted** policy ships with a large number of port types. Per-service documentation on SELinux types, Booleans, and port types can be found in the service-specific SELinux man pages found in the *selinux-policy-devel* package.

If these man pages are not yet installed on your system, follow this procedure:

```
[root@server1 ~]# yum -y install selinux-policy-devel
[root@server1 ~]# mandb
[root@server1 ~]# man -k _selinux
```

Removing port labels

The syntax for removing a custom port label is the same as the syntax for adding a port label, but instead of using the **-a** option (for Add), the **-d** option (for Delete) is used.

For example, to remove the binding of port **71/TCP** to **gopher_port_t**:

```
[root@server1 ~]# semanage port -d -t gopher_port_t -p tcp 71
```

Modifying port bindings

If an administrator has accidentally assigned the wrong type to a port, or requirements have changed, it's possible to modify the label associated with a port. This is a more efficient process than removing the old binding and adding a new one. Modifications require the **-m** option.

For example, to modify port **71/TCP** from **gopher_port_t** to **http_port_t**, an administrator can use the following command:

```
[root@server1 ~]# semanage port -m -t http_port_t -p tcp 71
```

References

semanage(8), **semanage-port**(8), ***_selinux**(8), and **system-config-selinux**(8) man pages

Practice: Managing SELinux Port Labeling

Guided exercise

In this lab, you will configure your **server1** system to allow **http** access on a nonstandard port.

Resources:	
Machines:	**desktop1** and **server1**

Outcomes:
A web server running on **server1** successfully serving content on a nonstandard port.

Before you begin...
- Reset your **server1** system.
- Log into your **server1** system and break the web server.

```
[root@server1 ~]# sed -i "s/Listen 80/Listen 82/" /etc/httpd/conf/httpd.conf
[root@server1 ~]# echo "Hello" > /var/www/html/index.html
```

Your organization is deploying a new custom web application. Unfortunately for you, the web application is running on a nonstandard port; in this case, **82/TCP**.

One of your developers has already configured the application on your **server1**. But, not being a rock star system administrator, he failed in getting the web server to start successfully. Your mission, if you choose to accept it, is to get the **httpd.service** service on **server1** successfully started, and serving out content to your **desktop1** system over port **82/TCP**.

1. Start by restarting the **httpd.service**.

   1.1.
   ```
   [root@server1 ~]# systemctl restart httpd.service
   Job for httpd.service failed. See 'systemctl status httpd.service' and
   'journalctl -xn' for details
   ```

   1.2. View the output from **systemctl status -l httpd.service**.

   ```
   [root@server1 ~]# systemctl status -l httpd.service
   ...
     Permission denied: AH00072: make_sock: could not bind to address 0.0.0.0:82
   ...
   ```

   1.3. Check if SELinux is blocking **httpd** from binding to port **82/TCP**.

   ```
   [root@server1 ~]# sealert -a /var/log/audit/audit.log
   ```

2. Configure SELinux to allow **httpd** to bind to port **82/TCP**, then restart the **httpd.service** service.

   2.1. Use **semanage** to find an appropriate port type for port **82/TCP**.

```
[root@server1 ~]# semanage port -l | grep http
```

http_port_t seems promising, since it is what the normal **http** port (**80/TCP**) is also assigned to.

2.2. Assign port **82/TCP** the **http_port_t** type.

```
[root@server1 ~]# semanage port -a -t http_port_t -p tcp 82
```

2.3. Restart the **httpd.service** service.

```
[root@server1 ~]# systemctl restart httpd.service
```

3. Check if you can now access the web server running on port **82/TCP**.

-
```
[root@server1 ~]# curl http://server1.example.com:82
Hello
```

4. Check if you can access the new web service from your **desktop1** system.

-
```
[root@desktop1 ~]# curl http://server1.example.com:82
curl: (7) Failed to connect to server1.example.com:82; No route to host
```

That error means you still can't connect from **desktop1**. Take a minute to think up some probable causes for this failure.

5. On your **server1** system, open up port **82/TCP** on your firewall.

5.1. Open port **82/TCP** in the permanent configuration for the default zone on the firewall on **server1**.

```
[root@server1 ~]# firewall-cmd --permanent --add-port=82/tcp
```

5.2. Activate your firewall changes on **server1**.

```
[root@server1 ~]# firewall-cmd --reload
```

6. Check if you can now access the new web service from your **desktop1** system.

-
```
[root@desktop1 ~]# curl http://server1.example.com:82
Hello
```

Summary

Managing Rich Rules

In this section, students learned how to configure more complex firewall configurations using firewalld's support for "rich language rules."

Masquerading and Port Forwarding

In this section, students learned how to describe and implement Network Address Translation (NAT).

Managing SELinux Port Labeling

In this section, students learned how to ensure network ports have the correct SELinux type so that services are able to bind to them.

CHAPTER 4

MANAGING DNS FOR SERVERS

Overview	
Goal	To set and verify correct DNS records for systems and configure secure caching DNS name service.
Objectives	• Explain how DNS is used to resolve names and addresses and the purpose of key DNS resource records. • Configure *unbound* to act as a secure local caching nameserver. • Describe common DNS configuration problems and how to identify and resolve them.
Sections	• DNS Concepts (and Practice) • Configuring a Caching Nameserver (and Practice) • DNS Troubleshooting (and Practice)
Lab	• Managing DNS for Servers

DNS Concepts

Objectives

After completing this section, students should be able to:

- Explain the hierarchical structure of the Domain Name System (DNS).
- Differentiate between domains, subdomains, and zones.
- Identify the differences between different resource record types.

The Domain Name System

The *Domain Name System (DNS)* is a hierarchical naming system that serves as a directory of networked hosts and resources. Information in the directory maps network names to data and is maintained in logical entries known as resource records. The DNS hierarchy begins with the *root* domain "**.**" at the top and branches downward to multiple next-level domains.

Each level of the DNS hierarchy is delineated by the "." in domain names, with "**.**" as the top level. Domains such as **com**, **net**, and **org** occupy the second level of the hierarchy and domains such as **example.com** and **redhat.com** occupy the third level and so on.

When working with DNS, it is important to clarify some of the common terms used to refer to the structure of the DNS hierarchy, such as **domain**, **subdomain**, and **zone**.

Domain

A **domain** is a collection of resource records that ends in a common name and represents an entire subtree of the DNS name space, such as **example.com**. The largest possible domain is the *root* domain, "**.**", which includes the whole DNS namespace.

A *top-level domain (TLD)* is a domain that has only one component. *Generic TLDs (gTLDs)* were originally organized by theme, and include **.com**, **.edu**, **.net**, etc. *Country code TLDs (ccTLDs)* are organized on a national basis, and include **.us**, **.uk**, **.cn**, **.ru**, etc.

Subdomain

A **subdomain** is a domain that is a subtree of another domain. This term is used when discussing the relationship of two domains to each other. For example, **lab.example.com** is a subdomain of **example.com**.

Zone

A **zone** is the portion of a domain for which a particular nameserver is directly responsible, or authoritative. This may be an entire domain, or just part of a domain with some or all of its subdomains delegated to other nameserver(s).

Anatomy of DNS lookups

When a system needs to perform name resolution using a DNS server, it begins by sending queries to the servers listed in **/etc/resolv.conf** in order, until it gets a response or runs out of servers. The **host** or **dig** commands can be used to manually look up DNS names.

Local authoritative data

When the query arrives at a DNS server, the server first determines whether the information being queried resides in a zone that it is authoritative for. If the server is an authority for the

zone that the name or address being queried belongs to, then the server responds to the client with the information contained in its local zone file. This type of response is referred to as an *authoritative answer (aa)*, since the server providing the response is authoritative for the data provided. Authoritative answers from a nameserver have the **aa** flag turned on in the header of the DNS response.

Local cached non-authoritative data
If the DNS server is not an authority for the record in question, but has recently obtained the record to answer a previous query, it may still have a copy of the record in its cache. The cache is where answers to queries are stored for a specified time, determined by a value contained in every resource record response called the *Time To Live (TTL)*. If an answer exists in the server's cache, it is provided to the client. This answer will not have the **aa** flag set, since the server is not authoritative for the data being provided.

Remote non-authoritative data via recursion
If the DNS server is not authoritative for the name being queried, and it does not possess the record in its cache, it will then attempt to retrieve the record via an iterative process known as recursion. A DNS server with an empty cache begins the recursion process by querying one of the root nameservers by IP address retrieved from its local, pre-populated *root hints file*. The root nameserver will then likely respond with a referral, which indicates the nameservers that are authoritative for the TLD that contains the name being queried.

Upon receiving the referral, the DNS server will then perform another iterative query to the TLD authoritative nameserver it was referred to. Depending on whether there are further remaining delegations in the name being queried, this authoritative nameserver will either send an authoritative answer or yet another referral. This continues until an authoritative server is reached and responds with an authoritative answer.

The final answer, along with all the intermediate answers obtained prior to it, are cached by the DNS server to improve performance. If during a lookup for **www.example.com** the DNS server finds out that the **example.com** zone has authoritative nameservers, it will query those servers directly for any future queries for information in the **example.com** zone, rather than starting recursion again at the root nameservers.

DNS resource records

DNS *resource records (RRs)* are entries in a DNS zone that specify information about a particular name or object in the zone. A resource record contains a **type**, a **TTL**, a **class**, and **data** elements organized in the following format:

```
owner-name          TTL     class     type     data
www.example.com.    300     IN        A        192.168.1.10
```

Resource Record Fields

Field name	Content
owner-name	The name for this resource record.
TTL	The *Time To Live* of the resource record in seconds. This specifies how long this resource record should be cached by DNS resolvers.
class	The "class" of the record, almost always **IN** ("Internet").
type	The type indicates the sort of information stored by this record. For example, an **A** record maps a host name to an IPv4 address.

Field name	Content
data	The data stored by this record. The exact format varies by record type.

There are a number of important resource record types:

A (IPv4 address) records
An **A** resource record maps a host name to an IPv4 address.

```
[student@server1 ~]$ host -v -t A example.com
Trying "example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22681
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;example.com.   IN A

;; ANSWER SECTION:
example.com.  86400 IN A 192.168.0.254

Recieved 96 bytes from 192.168.0.254#53 in 1 ms
```

AAAA (IPv6 address) records
An **AAAA** resource record ("quad-A" record) maps a host name to an IPv6 address.

```
[student@server1 ~]$ host -v -t AAAA a.root-servers.net
Trying "a.root-servers.net"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18194
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 13, ADDITIONAL: 12

;; QUESTION SECTION:
;a.root-servers.net.  IN AAAA

;; ANSWER SECTION:
a.root-servers.net. 604800 IN AAAA 2001:503:ba3e::2:30

Received 64 bytes from 192.168.0.254#53 in 78 ms
```

CNAME (canonical name) records
A **CNAME** resource record aliases one name to another name (the *canonical name*), which should have **A** or **AAAA** records.

When a DNS resolver receives a **CNAME** record in response to a query, it will reissue the query using the canonical name instead of the original name.

The data field of **CNAME** records can point to a name anywhere in DNS, whether internal or external to the zone:

```
www-dev.example.com. IN CNAME lab.example.com.
www.example.com.     IN CNAME www.redhat.com.
```

CNAME records are useful, but should be used with some care. In general, pointing a **CNAME** records to other **CNAME** records should be avoided for efficiency and fragility reasons and to avoid creating a **CNAME** loop by accident. The chain of **CNAME** record must end in **A** and/or **AAAA** records. Note that there are legitimate uses for **CNAME** chains when using Content Delivery Networks (CDNs) to improve the speed and reliability of data delivery over the Internet. Likewise,

NS and **MX** records must not be pointed at **CNAME** records but at names with **A** and/or **AAAA** records.

```
[student@server1 ~]$ host -v -t A ipa-ca.server0.example.com
Trying "ipa-ca.server0.example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11931
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;ipa-ca.server0.example.com. IN A

;; ANSWER SECTION:
ipa-ca.server0.example.com. 86400 IN CNAME server0.example.com.
server0.example.com. 86400 IN A 192.168.0.101

Recieved 125 bytes from 192.168.0.254#53 in 1 ms
```

PTR (pointer) records

A **PTR** record maps IPv4 or IPv6 addresses to a host name. They are used for *reverse DNS resolution*.

PTR records code the IP address in a special format that acts like a host name. For IPv4 addresses, the address is reversed, most specific part first, and the result is treated as a host in a subdomain of the special domain in-addr.arpa. For IPv6 addresses, the address is split into subdomains on nibble boundaries (every hexadecimal digit) and set up as a subdomain of the special domain ip6.arpa, as seen in the following example. While this syntax may seem strange, it makes it simpler for DNS administrators to delegate responsibility for ranges of addresses to other DNS administrators.

```
[student@server1 ~]$ host -v -t PTR 192.168.0.1
Trying "1.0.168.192.in-addr.arpa"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36389
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;1.0.168.192.in-addr.arpa. IN PTR

;; ANSWER SECTION:
1.0.168.192.in-addr.arpa. 86400 IN PTR desktop0.example.com.

Received 127 bytes from 192.168.0.254#53 in 2 ms
```

```
[student@server1 ~]$ host -v -t PTR 2001:503:ba3e::2:30
Trying "0.3.0.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.3.a.b.3.0.5.0.1.0.0.2.ip6.arpa"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32138
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;0.3.0.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.3.a.b.3.0.5.0.1.0.0.2.ip6.arpa. IN PTR

;; ANSWER SECTION:
0.3.0.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.3.a.b.3.0.5.0.1.0.0.2.ip6.arpa. 86400 IN PTR
 a.root-servers.net.

Received 122 bytes from 192.168.0.254#53 in 174 ms
```

NS (name server) records

An **NS** record maps a domain name to a DNS name server which is authoritative for its DNS zone.

Every public authoritative name server for the zone must have an **NS** record.

```
[student@server1 ~]$ host -v -t NS example.com
Trying "example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29362
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; QUESTION SECTION:
;example.com.   IN NS

;; ANSWER SECTION:
example.com.  86400 IN NS classroom.example.com.

Received 80 bytes from 192.168.0.254#53 in 0 ms
```

SOA (start of authority) records

An **SOA** record provides information about how a DNS zone works.

There will be exactly one **SOA** record for a zone. It specifies which of the zone's name servers is the primary one (the *master*), information on how secondary (*slave*) name servers should update their copy of the information, and the zone's management contact. Its data field contains the following elements:

SOA record data elements

Data element	Content
Master nameserver	The host name of the nameserver which is the original source of domain information, and which may accept dynamic DNS updates if the zone supports them.
RNAME	The email address of the person responsible for the DNS zone (the *hostmaster*). The **@** in the email address is replaced with a "**.**" in the RNAME. For example, an email address of **hostmaster@example.com** is written as **hostmaster.example.com**.
Serial number	The version number of the zone, which is increased when there is any change to zone records.
Refresh	How frequently the slave servers should check for zone updates, in seconds.
Retry	How long a slave server should wait before retrying a failed refresh attempt, in seconds.
Expiry	If refreshes have been failing, how long a slave server should wait before it stops using its old copy of the zone to respond to queries, in seconds.
Minimum	If a resolver looks up a name and it does not exist (gets a *nonexistent domain (NXDOMAIN)* response), how long it should cache the information that the record does not exist, in seconds.

```
[student@server1 ~]$ host -v -t SOA example.com
Trying "example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58434
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```

```
;; QUESTION SECTION:
;example.com.   IN SOA

;; ANSWER SECTION:
example.com.  86400 IN SOA classroom.example.com. root.classroom.example.com. 2013091600
 3600 300 604800 60

Received 121 bytes from 192.168.0.254#53 in 0 ms
```

MX (mail exchange) records

An **MX** record maps a domain name to a *mail exchange* which will accept email for that name.

The data for this record type is a preference number (lowest preferred) used to determine the order in which to pick between multiple **MX** records, and a host name for a mail exchange for that name.

```
[student@server1 ~]$ host -v -t MX example.com
Trying "example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47187
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;example.com.   IN MX

;; ANSWER SECTION:
example.com.  86400 IN MX 10 classroom.example.com.

Received 96 bytes from 192.168.0.254#53 in 0 ms
```

TXT (text) records

A **TXT** record is used to map a name to arbitrary human-readable text.

TXT records are commonly used to supply data used by *Sender Policy Framework (SPF)*, *DomainKeys Identified Mail (DKIM)*, *Domain-based Message Authentication, Reporting and Conformance (DMARC)*, and so on.

```
[student@server1 ~]$ host -v -t TXT lwn.net
Trying "lwn.net"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 41137
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;lwn.net.   IN TXT

;; ANSWER SECTION:
lwn.net.  28619 IN TXT "v=spf1 ip4:72.51.34.34 ip4:70.33.254.29 -all"

Received 638 bytes from 192.168.2.11#53 in 74 ms
```

SRV (service) records

An **SRV** record is used to locate the hosts which support a particular service for a domain.

Using a domain name formatted to include a service and a protocol name, *_service._protocol.domainname*, **SRV** records provide the names of the hosts that provide that service for the domain, as well as the port number that the service listens on. **SRV** records also include **priority** and **weight** values to indicate the order in which hosts should be used when multiple hosts are available for a particular service.

This example **SRV** record indicates that the **server0.example.com** domain provides the **LDAP** service using **TCP** on port **389** on host **server0.example.com** with a priority of **0** and a weighting of **100**.

```
[student@server1 ~]$ host -v -t SRV _ldap._tcp.server0.example.com
Trying "_ldap._tcp.server0.example.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35665
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 3

;; QUESTION SECTION:
;_ldap._tcp.server0.example.com. IN SRV

;; ANSWER SECTION:
_ldap._tcp.server0.example.com. 86400 IN SRV 0 100 389 server0.example.com.

Received 154 bytes from 192.168.0.254#53 in 0 ms
```

Hosts and resource records

A typical host, whether a client or a server, will have the following records:

- One or more **A** and/or **AAAA** records mapping its host name to its IP addresses
- A **PTR** record for each of its IP addresses, reverse mapping them to its host name
- Optionally, one or more **CNAME** records mapping alternate names to its canonical host name

A DNS zone will typically have, in addition to the records for the hosts in the zone:

- Exactly one **SOA** record to specify how the zone works
- An **NS** record for each of its authoritative name servers
- One or more **MX** records mapping the domain name to the mail exchange which receives email for addresses ending in the domain name
- Optionally, one or more **TXT** records for functions such as SPF or Google Site Verification
- Optionally, one or more **SRV** records to locate services in the domain

References

host(1) and **dig**(1) man pages

> RFC 1034: Domain names - concepts and facilities
> http://tools.ietf.org/html/rfc1034

> RFC 1035: Domain names - implementation and specification
> http://tools.ietf.org/html/rfc1035

> RFC 2181: Clarifications to the DNS Specification
> http://tools.ietf.org/html/rfc2181

Configuring a Caching Nameserver

Objectives

After completing this section, students should be able to configure a secure, caching nameserver using the **unbound** DNS server.

Caching nameservers and DNSSEC

Caching nameserver

Caching nameservers store DNS query results in a local cache and removes resource records from the cache when their TTLs expire. It is common to set up caching nameservers to perform queries on behalf of clients on the local network. This greatly improves the efficiency of DNS name resolutions by reducing DNS traffic across the Internet. As the cache grows, DNS performance improves as the caching nameserver answers more and more client queries from its local cache.

DNSSEC validation

Given the stateless nature of UDP, DNS transactions are prone to spoofing and tampering. Caching nameservers have historically been favored targets of attackers looking to redirect or hijack network traffic. This is often achieved by exploiting vulnerabilities in DNS server software to fool a DNS server into accepting and populating malicious data into its cache, a technique commonly referred to as *cache poisoning*. Once the attacker succeeds in poisoning a DNS server's cache, they effectively compromise the DNS data received by the numerous clients utilizing the caching name service on the DNS server and can consequently redirect or hijack the clients' network traffic.

While a caching nameserver can greatly improve DNS performance on the local network, they can also provide improved security by performing *Domain Name System Security Extensions (DNSSEC)* validation. DNSSEC validation enabled at the caching nameserver allows the authenticity and integrity of resource records to be validated prior to being placed in the cache for use by clients, and therefore protects clients against the consequences of cache poisoning.

Configuring and administering *unbound* as a caching nameserver

Several packages are available for configuring a caching nameserver, including *bind*, *dnsmasq*, and *unbound*. In this example, please follow along while the instructor demonstrates the configuration and administration of *unbound* as a secure, caching nameserver with DNSSEC validation enabled.

Configuring unbound

To configure **unbound** as a secure, caching nameserver:

1. Install **unbound**.

   As *root*, install the **unbound** package.

   ```
   [root@server1 ~]# yum install -y unbound
   ```

2. Start and enable **unbound.service**.

```
[root@server1 ~]# systemctl start unbound.service
[root@server1 ~]# systemctl enable unbound.service
ln -s '/usr/lib/systemd/system/unbound.service' '/etc/systemd/system/multi-
user.target.wants/unbound.service'
```

3. Configure the network interface to listen on.

   By default, **unbound** only listens on the **localhost** network interface. To make **unbound** available to remote clients as a caching nameserver, use the **interface** option in the **server** clause of **/etc/unbound/unbound.conf** to specify the network interface(s) to listen on. A value of **0.0.0.0** will configure **unbound** to listen on all network interfaces:

   ```
   interface: 0.0.0.0
   ```

4. Configure client access.

   By default, **unbound** refuses recursive queries from all clients. In the **server** clause of **/etc/unbound/unbound.conf**, use the **access-control** option to specify which clients are allowed to make recursive queries.

   ```
   access-control: 192.168.0.0/24 allow
   ```

5. Configure forwarding.

   In **/etc/unbound/unbound.conf**, create a **forward-zone** clause to specify which DNS server(s) to forward queries to. DNS servers can be specified by host name using the **forward-host** option, or by IP address using the **forward-addr** option. For a caching nameserver, forward all queries by specifying a **forward-zone** of "**.**".

   ```
   forward-zone:
     name: "."
     forward-addr: 192.168.0.254
   ```

6. If desired, bypass DNSSEC validation for select unsigned zones.

   By default, **unbound** is enabled to perform DNSSEC validation to verify all DNS responses received. The **domain-insecure** option in the **server** clause of **/etc/unbound/unbound.conf** can be used to specify a domain for which DNSSEC validation should be skipped. This is often desirable when dealing with an unsigned internal domain that would otherwise fail trust chain validation.

   ```
   domain-insecure: example.com
   ```

7. If desired, install trust anchors for select signed zones without complete chain of trust.

   Since not all ccTLDs have completed implementation of DNSSEC, the subdomains of these ccTLDs can be DNSSEC-signed but still have a broken *chain of trust*. This problem can be overcome by using the **trust-anchor** option in the **server** clause of **/etc/unbound/unbound.conf** to specify a trust anchor for the zone. Obtain the **DNSKEY** record for the *key*

signing key (KSK) of the zone using **dig** and input it as the value for the **trust-anchor** option.

```
[student@server1 ~]$ dig +dnssec DNSKEY example.com
```

```
trust-anchor: "example.com. 3600 IN DNSKEY 257 3 8 AwEAAawt7HplI5M8GGAsxuyCyjF0l
+QlcgVN11CRZ4vP66qbDCX0BnSh Z11BGb//4zSG/8mmBHirL2FLg+mVuIIxig
+iroZYjh4iTKVOhv2hZftR wyrQHK++qXvCCWN3ki51RG/e8R4kOEV71rZ8OgQvPWx6F91qroqOPpcf
 7PPxippeHOn+PxnP0hpyLyo1mx1rPs/cMpL3jOMufGP+LJYh+fBU7lt0
 sP5i09HaJPruzyZML9BPtpv8ZAdQhwtXVG0+MnET2qT/1+TljpxZn6ye egFRCFRHBjMo6iiRJnUWra/
klkrgEn2Q+BXGTOMTTKQdYz4OxYEa1z7a pu3a09dYNBM="
```

8. Save changes to **/etc/unbound/unbound.conf**.

9. Check the **/etc/unbound/unbound.conf** configuration file for syntax errors.

```
[root@server1 ~]# unbound-checkconf
unbound-checkconf: no errors in /etc/unbound/unbound.conf
```

10. Restart **unbound.service**.

```
[root@server1 ~]# systemctl restart unbound.service
```

11. Configure the firewall to allow DNS traffic.

```
[root@server1 ~]# firewall-cmd --permanent --add-service=dns
success
[root@server1 ~]# firewall-cmd --reload
success
```

Dumping and loading unbound cache

Administrators of caching nameservers need to dump out cache data when troubleshooting DNS issues, such as those resulting from stale resource records. With an **unbound** DNS server, the cache can be dumped by running the **unbound-control** utility in conjunction with the **dump_cache** subcommand.

```
[root@server1 ~]# unbound-control dump_cache
START_RRSET_CACHE
;rrset 86395 1 0 3 3
classroom.example.com. 86395 IN A 192.168.0.254
;rrset 86395 1 0 7 3
example.com. 86395 IN NS classroom.example.com.
;rrset 86395 1 0 8 3
example.com. 86395 IN A 192.168.0.254
END_RRSET_CACHE
START_MSG_CACHE
msg example.com. IN A 33152 1 86395 3 1 1 1
example.com. IN A 0
example.com. IN NS 0
classroom.example.com. IN A 0
END_MSG_CACHE
EOF
```

Executing **unbound-control** with the **dump_cache** command dumps out the cache to *stdout* in a text format. This output can be directed to a file for storage and be loaded back into cache later with **unbound-control load_cache**, if desired. **unbound-control load_cache** reads from *stdin* to populate the cache.

```
[root@server1 ~]# unbound-control load_cache < dump.out
ok
```

Flushing unbound cache

Administrators of caching nameservers also need to purge outdated resource records from cache from time to time. Erroneous and outdated resource records in cache will keep their newly corrected counterparts from becoming available to clients until the TTLs on the outdated resource records expire. Rather than waiting for TTL expiration, administrators can forcibly purge the outdated records from cache by executing **unbound-control** with the **flush** subcommand.

```
[root@server1 ~]# unbound-control flush www.example.com
ok
```

If all resource records belonging to a domain need to be purged from the cache of an **unbound** DNS server, **unbound-control** can be executed with the **flush_zone** subcommand.

```
[root@server1 ~]# unbound-control flush_zone example.com
ok removed 3 rrsets, 1 messages and 0 key entries
```

Updating Local Caching unbound Configuration with dnssec-trigger

In addition to providing caching name service for a local subnet, **unbound** can also be useful as a local caching nameserver to provide secure DNS name resolution for local use on an individual system. For a local caching nameserver setup, the **nameserver** entry in **/etc/resolv.conf** will be configured to point to localhost where **unbound** is listening. The **unbound** configuration will forward DNS requests to upstream nameservers and validate their responses.

For DHCP systems running local caching name service, the upstream nameservers specified in **unbound**'s configuration may become outdated if DNS servers provided by DHCP change. The **dnssec-trigger** tool supplied by the package of the same name can be leveraged to automatically update forwarder settings in **unbound**'s configuration file to point to the new DNS servers. The use of the **dnssec-trigger** tool in conjunction with **unbound** is mostly useful for secure DNS name resolution on roaming client machines.

References

unbound(8), **unbound-checkconf**(8), **unbound.conf**(5), **unbound-control**(8), **dnssec-trigger**(8) man pages

Practice: Configuring unbound as a Caching Nameserver

Guided exercise

In this lab, you will configure **unbound** as a caching nameserver and administer its cache data.

Resources:	
Files:	/etc/unbound/unbound.conf
Machines:	• desktop1 • server1

Outcomes:
The **unbound** service enabled and configured as a caching nameserver on **server1** to listen on interface **192.168.0.101** and to accept connections from **192.168.0.0/24**. Recursive queries are forwarded to **192.168.0.254**. The **example.com** zone is exempted from DNSSEC validation.

Before you begin...
- Reset the **server1** system.
- Log into the **desktop1** system.
- Log into the **server1** system and switch to root using **su**.

You have been asked to improve name resolution performance and enhance DNS security in your company's datacenter. You have decided to deploy a caching nameserver using an **unbound** DNS server. You will configure **unbound** to respond only to queries on the datacenter subnet, **192.168.0.0/24**. Recursive queries will be forwarded to the company's main internal DNS server at **192.168.0.254**. This main internal DNS server hosts an internal, split DNS copy of the company's **example.com** zone. This **example.com** zone is not DNSSEC-signed, so it will need to be exempted from DNSSEC validation on your caching nameserver. After configuration of the caching nameserver is complete on **server1**, you will test it by querying for the host names **desktop1.example.com** and **server1.example.com** from **desktop1**. Verify that the queries made are populated into the nameserver's cache. Lastly, you will purge **server1.example.com** from the cache.

1. Install the *unbound* package on **server1**.

```
[root@server1 ~]# yum install -y unbound
```

2. Enable and start the **unbound** service.

```
[root@server1 ~]# systemctl enable unbound.service
ln -s '/usr/lib/systemd/system/unbound.service' '/etc/systemd/system/multi-
user.target.wants/unbound.service'
[root@server1 ~]# systemctl start unbound.service
```

3. Configure **unbound** to allow queries from the **192.168.0.0/24** subnet, exempt the **example.com** zone from DNSSEC validation, and forward all queries to **192.168.0.254**.

   3.1. By default, **unbound** listens on the loopback interface. Configure **unbound** to listen on the 192.168.0.101 interface on **server1** by adding the following option in the **server** clause of **/etc/unbound/unbound.conf**.

   ```
   interface: 192.168.0.101
   ```

   3.2. Allow queries from the **192.168.0.0/24** subnet by adding the following option in the **server** clause of **/etc/unbound/unbound.conf**.

   ```
   access-control: 192.168.0.0/24 allow
   ```

   3.3. Exempt the **example.com** zone from DNSSEC validation by adding the following option in the **server** clause of **/etc/unbound/unbound.conf**.

   ```
   domain-insecure: "example.com"
   ```

   3.4. Forward all queries to **192.168.0.254** by adding a **forward-zone** clause to the end of the **/etc/unbound/unbound.conf** file.

   ```
   forward-zone:
              name: .
              forward-addr: 192.168.0.254
   ```

4. Check **/etc/unbound/unbound.conf** for syntax errors.

   ```
   [root@server1 ~]# unbound-checkconf
   unbound-checkconf: no errors in /etc/unbound/unbound.conf
   ```

5. Restart the **unbound** service.

   ```
   [root@server1 ~]# systemctl restart unbound.service
   ```

6. Configure the firewall to allow DNS traffic.

   ```
   [root@server1 ~]# firewall-cmd --permanent --add-service=dns
   success
   [root@server1 ~]# firewall-cmd --reload
   success
   ```

7. Verify the caching name service by performing queries and examining the contents of the cache.

   7.1. Dump the cache to see its contents.

   ```
   [root@server1 ~]# unbound-control dump_cache
       START_RRSET_CACHE
       END_RRSET_CACHE
   ```

```
    START_MSG_CACHE
    END_MSG_CACHE
    EOF
```

7.2. From **desktop1**, query **server1** for the **A** record of host name **desktop1.example.com**.

```
[student@desktop1 ~]$ dig @server1.example.com A desktop1.example.com
...
desktop1.example.com. 86349 IN A 192.168.0.1
...
```

7.3. From **desktop1**, query **server1** for the **A** record of host name **server1.example.com**.

```
[student@desktop1 ~]$ dig @server1.example.com A server1.example.com
...
server1.example.com. 86364 IN A 192.168.0.101
...
```

7.4. On **server1**, dump out the cache again. You should see the queried records in the cache.

```
[root@server1 ~]# unbound-control dump_cache
START_RRSET_CACHE
;rrset 85886 1 0 8 3
server1.example.com. 85886 IN A 192.168.0.101
;rrset 85878 1 0 8 X
desktop1.example.com. 85878 IN A 192.168.0.1
;rrset 85878 1 0 7 3
example.com. 85878 IN NS instructor.example.com.
;rrset 85878 1 0 3 3
instructor.example.com. 85878 IN A 192.168.0.254
END_RRSET_CACHE
START_MSG_CACHE
msg desktop1.example.com. IN A 33152 1 85878 3 1 1 1
desktop1.example.com. IN A 0
example.com. IN NS 0
instructor.example.com. IN A 0
msg server1.example.com. IN A 33152 1 85886 3 1 1 1
server1.example.com. IN A 0
example.com. IN NS 0
instructor.example.com. IN A 0
END_MSG_CACHE
EOF
```

7.5. Purge the **server1.example.com** record from the cache.

```
[root@server1 ~]# unbound-control flush server1.example.com
ok
```

7.6. On **server1**, dump out the cache again. You should no longer see the **A** record for **server1.example.com** in the cache.

```
[root@server1 ~]# unbound-control dump_cache
```

```
START_RRSET_CACHE
;rrset 85878 1 0 8 X
desktop1.example.com. 85878 IN A 192.168.0.1
;rrset 85878 1 0 7 3
example.com. 85878 IN NS instructor.example.com.
;rrset 85878 1 0 3 3
instructor.example.com. 85878 IN A 192.168.0.254
END_RRSET_CACHE
START_MSG_CACHE
msg desktop1.example.com. IN A 33152 1 85878 3 1 1 1
desktop1.example.com. IN A 0
example.com. IN NS 0
instructor.example.com. IN A 0
msg server1.example.com. IN A 33152 1 85886 3 1 1 1
server1.example.com. IN A 0
example.com. IN NS 0
instructor.example.com. IN A 0
END_MSG_CACHE
EOF
```

DNS Troubleshooting

Objectives

After completing this section, students should be able to:

- Use **dig** to troubleshoot common DNS problems.
- Identify symptoms and causes associated with common DNS issues.

Troubleshooting DNS

Due to the client-server architecture of DNS, properly working DNS name resolution on a system is almost always dependent on not only the proper configuration and operation of DNS on that system, but also on that of its resolving nameserver and the many authoritative nameservers used to resolve its DNS requests. Since DNS is a distributed directory , recursive name resolution often involves numerous behind-the-scenes interactions with many different authoritative nameservers. These numerous interactions create many possible points for failure.

The use of caching nameservers significantly reduces DNS workloads and improves DNS performance. However, the caching function adds another point of failure by creating scenarios where it is possible for DNS responses received by clients to be inaccurate due to the data no longer being current.

Due to the critical role that DNS plays in the functioning of network services, it is important that Linux administrators be able to quickly resolve DNS issues when they occur, in order to minimize service interruptions. The key to accurate and efficient DNS troubleshooting is being able to pinpoint which of the multiple points in the myriad of behind-the-scenes client-server interactions is responsible for the unexpected behavior observed. This requires the use of proper tools and a clear understanding of the diagnostic data they provide. *Domain Internet Groper (dig)* is a good tool for investigating DNS issues due to its verbose diagnostic output.

Name resolution methods

Because DNS service is often the most widely used method of name resolution, it often bears the blame whenever unexpected name resolution results occur. Remember that aside from DNS, in a heterogeneous environment, name resolution on networked hosts can occur via other methods, such as local hosts files, Windows Internet Name Service (WINS), etc.

On Linux systems, name resolution is attempted first with the hosts file **/etc/hosts** by default, per order specified in **/etc/nsswitch.conf**. Therefore, when beginning name resolution troubleshooting, do not leap to the assumption that the issue resides with DNS. Begin first by identifying the name resolution mechanism which is in play, rather than simply starting with DNS. The **getent** command from the *glibc-common* package, as well as the **gethostip** command from the *syslinux* package, can both be used to perform name resolution, mirroring the process used by most applications in following the order of host name resolution as dictated by **/etc/nsswitch.conf**.

```
[student@desktop1 ~]$ getent hosts example.com
192.168.0.254  example.com
```

```
[student@desktop1 ~]$ gethostip example.com
example.com 192.168.0.254 AC19FEFE
```

If the result of **getent** or **gethostip** differs from that produced by **dig**, then it's a clear indication that something besides DNS is responsible for the unexpected name resolution result. Consult **/etc/nsswitch.conf** to determine what other name resolution mechanisms are employed before DNS.

Client-server network connectivity

For DNS name resolution to work properly, a system must first be able to conduct client-server interactions with its resolving nameserver or other authoritative nameservers. Some common DNS issues that have their origin at this layer are the result of resolver and firewall misconfigurations.

When using **dig** to troubleshoot a DNS issue, if a response is not received from a DNS server, it is a clear indication that the cause lies with the client-server network connectivity to the DNS server.

```
[student@desktop1 ~]$ dig A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> A example.com
;; global options: +cmd
;; connection timed out; no servers could be reached
```

A possible cause is the inability to reach the DNS server due to incorrect DNS server IP address(es) in a system's DNS configuration. This could be in **/etc/resolv.conf** in the case of a system acting as a DNS client or in the **forward-zone** clause of **/etc/unbound/unbound.conf** in the case of a system configured as an **unbound** caching nameserver.

Another possible cause are firewall rules, on either the client or server system, blocking DNS traffic on port 53. While DNS mostly uses the UDP protocol, it is important to note that when response data sizes exceed 512 bytes, or 4096 bytes in the case of DNS servers that support *Extension Mechanism for DNS (EDNS)*, resolvers will fall back to using TCP to retry the query. Therefore, proper DNS configuration should allow for DNS traffic on port 53 for both UDP and TCP. Allowing port 53 traffic for UDP only will result in a truncation error when the resolver encounters a response that is larger than what it can handle over UDP.

```
[student@desktop1 ~]$ dig @server1.example.com A labhost1.example.com
;; Truncated, retrying in TCP mode.
;; Connection to 192.168.0.101#53(192.168.0.101) for labhost1.example.com failed:
host unreachable.
```

dig's **tcp** or **vc** options are helpful for troubleshooting whether DNS queries can succeed with TCP. These options force **dig** to use TCP, rather than the default behavior of using UDP first and falling back to TCP only when response size necessitates it.

```
[student@desktop1 ~]$ dig +tcp A example.com
```

When dealing with DNS issues at the network layer, **dig** provides very sparse output and it is therefore often useful to also use a network packet analyzer, such as **tcpdump**, to determine what is transpiring behind the scenes at the network layer. Using **tcpdump**, the administrator can determine information that cannot be ascertained with **dig** alone, such as the destination IP address of the DNS request, if request packets leave the client, if request packets reach the server, if response packets leave the server, if response packets reach the client, etc.

DNS response codes

If DNS client-server communication is successful, **dig** will generate much more verbose output detailing the nature of the response received from the DNS server. The **status** field in the

HEADER section of **dig**'s output reports the response code generated by the DNS server in response to the client's query.

```
[student@desktop1 ~]$ dig A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> A example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 30523
...
```

A status of **NOERROR** indicates the query was resolved successfully. If the server encounters problems fulfilling the client's query, one of the following common error statuses will be displayed.

DNS Return Codes

Code	Meaning
SERVFAIL	The nameserver encountered a problem while processing the query.
NXDOMAIN	The queried name does not exist in the zone.
REFUSED	The nameserver refused the client's DNS request due to policy restrictions.

SERVFAIL

A common cause of **SERVFAIL** status is the failure of the DNS server to communicate with the nameservers authoritative for the name being queried. This may be due to the authoritative nameservers being unavailable. It could also be a problem at the network layer interfering with the client-server communication between the DNS server and the authoritative nameservers, such as network routing issues or firewall rules at any hop in the network path.

To determine why a nameserver is generating a **SERVFAIL** status while performing recursion on behalf of a client's query, the administrator of the nameserver will need to determine which nameserver communication is causing the failure. **dig**'s **+trace** option is helpful in this scenario to see the results of nameserver's iterative queries starting with the root nameservers.

NXDOMAIN

An **NXDOMAIN** status indicates that no records were found associated with the name queried. If this is not the expected result and the query is directed at a server that is not authoritative for the name, then the server's cache may contain a negative cache for the name. The user can either wait for the server to expire the negative cache of that name, or submit a request to the server administrator to flush the name from the server's cache. Once the name is removed from cache, the server will query the authoritative nameserver to receive current resource records for the name.

The other scenario where an **NXDOMAIN** status may be unexpectedly encountered is when querying a **CNAME** record containing an *orphaned CNAME*. In a **CNAME** record, the name on the right side of the record, the canonical name, should point to a name that contains either **A** or **AAAA** records. If these associated **A** or **AAAA** records are nonexistent or are later removed, then the canonical name in the **CNAME** record is orphaned. When this occurs, queries for the owner name in the **CNAME** record will no longer be resolvable and will result in an **NXDOMAIN** return code.

REFUSED

A **REFUSED** status indicates that the DNS server has a policy restriction which keeps it from fulfilling the client's query. Policy restrictions are often implemented on DNS servers to restrict

which clients can make recursive queries and zone transfer requests. Some common causes of an unexpected **REFUSED** return code are clients configured to query the wrong DNS servers or DNS server misconfiguration causing valid client requests to be refused.

Other common DNS issues

Outdated cached data

A DNS return code of **NOERROR** signifies that no errors were encountered in resolving a query. However, it does not guarantee that there are no DNS issues present. There are situations where the DNS records in the DNS response may not match the expected result. The most common cause for an incorrect answer is that the answer originated from cached data, which is no longer current.

In these situations, first confirm that the response is indeed nonauthoritative cached data. This can be easily determined by looking at the **flags** section of **dig**'s output. DNS responses, which are authoritative answers, will be indicated by the presence of the **aa** flag.

```
[student@desktop1 ~]$ dig A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> A example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22257
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
```

DNS responses that originate from cached data are not authoritative and therefore will not have the **aa** flag set. The other telltale sign that an answer is coming from cache is the counting down of the resource record's TTL value in the responses of each subsequent query. TTLs of cached data will continuously count down to expiration while TTLs of authoritative data will always remain static.

Responses for nonexistent records

If a record has been removed from a zone and a response is still received when querying for the record, first confirm that the queries are not being answered from cached data. If the answer is authoritative as confirmed by the presence of the **aa** flag in **dig**'s output, then a possible cause is the presence of a *wildcard (*)* record in the zone.

```
*.example.com. IN A 192.168.0.254
```

A wildcard record serves as a catchall for all queries of a given type for a nonexistent name. With the previous wildcard record in place, if an **A** record previously existed for **server1.example.com** and it is removed, queries for the name will still succeed and the IP address in the wildcard **A** record will be provided in its place.

Non-FQDN name errors

In a zone file, host names which are not expressed as *Fully Qualified Domain Names (FQDNs)* are automatically expanded to FQDNs by appending the name of the zone. To indicate that a name is an FQDN in a zone file, it must be ended with a "**.**", i.e., "**www.example.com.**". Failure to do so can lead to different issues depending on the type of record that the mistake is made in. For example, such a mistake made in the type-specific data portion of **NS** records have the potential of incapacitating an entire zone, while a mistake made in **MX** records could cause a complete halt of email delivery for a domain.

Looping CNAME records

While technically feasible, **CNAME** records that point to **CNAME** records should be avoided to reduce DNS lookup inefficiency. Another reason this is undesirable is the possibility of creating unresolvable **CNAME** loops, such as:

```
test.example.com. IN CNAME lab.example.com.
lab.example.com.  IN CNAME test.example.com.
```

While a **CNAME** record with an orphaned **CNAME** will result in an **NXDOMAIN** status, looping **CNAME** records will return as **NOERROR**.

Missing PTR records

Many network services use DNS to perform reverse lookups of incoming connections from clients. The absence of **PTR** records in DNS may result in issues, the nature of which varies depending on the service. SSHD, by default, will perform reverse lookups of connecting client IPs. Absence of **PTR** records will lead to delays in the establishment of these connections.

Many MTAs incorporate reverse DNS lookups of connecting client IPs as a defense against malicious email clients. In fact, many MTAs are configured to reject client connections for IPs, which cannot be resolved with a **PTR** query in DNS. As such, administrators supporting network services need to ensure that they understand the requirements these services have for not just forward, but also reverse DNS lookups.

Round-robin DNS

A name can have multiple **A** or **AAAA** records in DNS. This is known as *round-robin DNS*, and is often used as a simple, low-cost, load-balancing mechanism to distribute network resource loads across multiple hosts. When a DNS client queries for a name that contains multiple **A** or **AAAA** records, all records are returned as a set. However, the order that the records are returned in the set permutates for each query. Since clients normally make use of the first address in the set, the variation in the order of the records in each response effectively results in a distribution of network service requests across the multiple IP addresses in these round-robin DNS records.

While round-robin DNS is a valid technical configuration, there are times when this configuration is inadvertently created. When a request to change the IP address of an **A** record is mistakenly implemented as a resource record addition rather than a resource record modification, then round-robin DNS is created. If the network resources on the old IP address is retired, the load distribution effect of the round-robin DNS will result in service failures for approximately half of the clients.

References

dig(1), **getent**(1), **gethostip**(1) man pages

Practice: Troubleshooting DNS

Guided exercise

In this lab, you will troubleshoot and resolve a name resolution issue by systematically verifying name service configurations to pinpoint the cause.

Resources:	
Files:	• /etc/nsswitch.conf • /etc/hosts • /etc/resolv.conf
Machines:	• server1

Outcomes:
The root cause for failed name resolution for **example.com** on **server1** will be identified and fixed. After the issue is resolved, name resolution for **example.com** on **server1** will succeed.

Before you begin...

- Reset the **server1** system.
- Log into and set up the **server1** system.

```
[student@server1 ~]$ wget -O - http://instructor.example.com/pub/server1-dns.sh | bash
```

A user reports that there is an issue occurring when an SSH session is initiated to **example.com** from **server1** and a "**Could not resolve hostname example.com: Name or service not known**" error is being generated. Therefore, the user complains that there is a problem with DNS name resolution of **example.com**. Like all other hosts on the network, **server1** should be using **192.168.0.254** for DNS resolution. You will troubleshoot the issue, identify the root cause, apply a fix, and then verify that the problem is resolved.

Note

You can not actually login to **example.com** using SSH. Seeing a login prompt is all that is required, and the login prompt can be cancelled by pressing **Ctrl**+**C**

1. Replicate the reported issue by attempting an SSH session to **example.com** from **server1**.

```
[student@server1 ~]$ ssh example.com
ssh: Could not resolve hostname example.com: Name or service not known
```

2. Verify the result of name resolution for **example.com**.

```
[student@server1 ~]$ getent hosts example.com
[student@server1 ~]$
```

3. Verify configurations which affect name resolution.

3.1. Verify the order that name services are used.

```
[student@server1 ~]$ grep ^hosts: /etc/nsswitch.conf
hosts:      files dns
```

3.2. Since **files** are used first, verify the contents of **/etc/hosts**.

```
[student@server1 ~]$ grep [[:space:]]example.com /etc/hosts
[student@server1 ~]$
```

3.3. Since no hosts file entry exist for **example.com**, verify the contents of **/etc/resolv.conf**; you should see that an incorrect nameserver IP is the cause of the name resolution failure.

```
[student@server1 ~]$ grep ^nameserver /etc/resolv.conf
nameserver 192.168.0.255
```

```
[student@server1 ~]$ dig @192.168.0.255 A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @192.168.0.255 A example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

4. As **root**, fix the error in **/etc/resolv.conf** and verify that this resolves the name resolution issue.

4.1. Since the **nameserver** entry populated by DHCP appears to have been manually modified, as root, force a refresh of the data from DHCP and validate that the entry is fixed.

```
[student@server1 ~]$ sudo systemctl restart NetworkManager
```

```
[student@server1 ~]$ grep ^nameserver /etc/resolv.conf
nameserver 192.168.0.254
```

4.2. Verify the results of name resolution for **example.com**.

```
[student@server1 ~]$ getent hosts example.com
192.168.0.254  example.com
```

```
[student@server1 ~]$ dig A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> example.com
;; global options: +cmd
;; Got answer:
;; -->>HEADER<<-- opcode: QUERY, status: NOERROR, id: 36048
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.com.   IN A
```

```
;; ANSWER SECTION:
example.com.  86400 IN A 192.168.0.254

;; AUTHORITY SECTION:
example.com.  86400 IN NS classroom.example.com.

;; ADDITIONAL SECTION:
classroom.example.com. 86400 IN A 192.168.0.254

;; Query time: 1 msec
;; SERVER: 192.168.0.254#53(192.168.0.254)
;; WHEN: Thu May 15 07:28:35 EDT 2014
;; MSG SIZE  rcvd: 96
```

4.3. Verify that SSH connection to **example.com** from **server1** now succeeds.

```
[student@server1 ~]$ ssh example.com
The authenticity of host 'example.com (192.168.0.254)' can't be established.
ECDSA key fingerprint is 12:b3:c8:3e:6b:d2:9f:43:67:a5:f2:2a:f0:7c:2f:b6.
Are you sure you want to continue connecting (yes/no)?
```

Lab: Managing DNS for Servers

Performance checklist

In this lab, you will use DNS troubleshooting techniques and knowledge of **unbound** configuration to resolve DNS issues with a misconfigured **unbound** caching nameserver.

Resources:	
Files:	• /etc/unbound/ubound.conf
Machines:	• desktop1 • server1

Outcomes:
The resolution of issues with DNS requests performed from **desktop1** against the **unbound** caching nameserver running on **server1**. Properly configured **unbound** caching nameserver running on **server1**.

Before you begin...
- Reset your **server1** system.
- Log into and set up your **server1** system.

```
[student@server1 -]$ wget -O - http://instructor.example.com/pub/server1-unbound.sh |
 bash
```

To improve DNS performance and security, a fellow system administrator has recently configured a secure caching nameserver running the **unbound** DNS server on **server1** to serve your local subnet, **192.168.0.0/24**. The caching nameserver uses DNS server **192.168.0.254** for name resolution. The administrator released the caching nameserver for use just prior to leaving for a week-long vacation.

Soon after, you receive reports of DNS issues from users trying to use the newly configured caching nameserver for the first time. One user provides a specific example and reports that name resolution for **example.com** fails.

You will conduct troubleshooting to determine and correct the **unbound** misconfiguration that is causing this issue.

1. On **desktop1**, replicate the reported issue to determine the nature and scope of the problem.
2. Verify the operation of the caching nameserver on **server1** by issuing queries to it locally from **server1**.
3. Fix the issue discovered and verify that it resolved the name resolution issue.
4. While the DNS communication now works, the return code in the DNS response indicates that there may be another misconfiguration issue present. Verify the **unbound** configuration on **server1**.

5. Fix the issue discovered and verify that it resolved the name resolution issue.

Solution

In this lab, you will use DNS troubleshooting techniques and knowledge of **unbound** configuration to resolve DNS issues with a misconfigured **unbound** caching nameserver.

Resources:	
Files:	• /etc/unbound/ubound.conf
Machines:	• desktop1 • server1

Outcomes:
The resolution of issues with DNS requests performed from **desktop1** against the **unbound** caching nameserver running on **server1**. Properly configured **unbound** caching nameserver running on **server1**.

Before you begin...

- Reset your **server1** system.
- Log into and set up your **server1** system.

```
[student@server1 -]$ wget -O - http://instructor.example.com/pub/server1-unbound.sh |
 bash
```

To improve DNS performance and security, a fellow system administrator has recently configured a secure caching nameserver running the **unbound** DNS server on **server1** to serve your local subnet, **192.168.0.0/24**. The caching nameserver uses DNS server **192.168.0.254** for name resolution. The administrator released the caching nameserver for use just prior to leaving for a week-long vacation.

Soon after, you receive reports of DNS issues from users trying to use the newly configured caching nameserver for the first time. One user provides a specific example and reports that name resolution for **example.com** fails.

You will conduct troubleshooting to determine and correct the **unbound** misconfiguration that is causing this issue.

1. On **desktop1**, replicate the reported issue to determine the nature and scope of the problem.
   - From **desktop1**, issue a query for the address of **example.com** to the caching nameserver on **server1**.

     ```
     [student@desktop1 -]$ dig @server1.example.com A example.com
       ; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @server1.example.com example.com
       ; (1 server found)
       ;; global options: +cmd
       ;; connection timed out; no servers could be reached
     ```

2. Verify the operation of the caching nameserver on **server1** by issuing queries to it locally from **server1**.

   2.1. Issue the query to the **localhost** interface. The query succeeds with no errors.

```
[student@server1 -]$ dig @localhost A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @localhost A example.com
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64095
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.com.   IN A

;; ANSWER SECTION:
example.com.  84526 IN A 192.168.0.254

;; Query time: 0 msec
;; SERVER: ::1#53(::1)
;; WHEN: Wed May 21 05:12:00 EDT 2014
;; MSG SIZE  rcvd: 56
```

2.2. Issue the query to the **192.168.0.101** interface. No response is received and the query times out.

```
[student@server1 -]$ dig @192.168.0.101 A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @192.168.0.101 A example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

2.3. Determine which interfaces **unbound** is listening on. You should discover that it's only listening on the **localhost** interface.

```
[student@server1 -]$ sudo ss -tulpn | grep -w 53
tcp    UNCONN     0      0              127.0.0.1:53                    *:*
 users:(("unbound",2192,5))
tcp    UNCONN     0      0                    ::1:53                   :::*
 users:(("unbound",2192,3))
tcp    LISTEN     0      5              127.0.0.1:53                    *:*
 users:(("unbound",2192,6))
tcp    LISTEN     0      5                    ::1:53                   :::*
 users:(("unbound",2192,4))
```

3. Fix the issue discovered and verify that it resolved the name resolution issue.

3.1. Configure **unbound** to listen on all interfaces by adding the following entry in **/etc/unbound/unbound.conf**.

```
interface: 0.0.0.0
```

3.2. Check the configuration for syntax errors.

```
[student@server1 -]$ sudo unbound-checkconf
unbound-checkconf: no errors in /etc/unbound/unbound.conf
```

3.3. Restart the service for the changes to take place.

```
[student@server1 -]$ sudo systemctl restart unbound
```

3.4. On **server1**, rerun the query to the **192.168.0.101** interface. You should now receive a response, but with a status of **REFUSED**.

```
[student@server1 -]$ dig @192.168.0.101 A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @192.168.0.101 A example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: REFUSED, id: 50719
;; flags: qr rd ad; QUERY: 0, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; Query time: 0 msec
;; SERVER: 192.168.0.101#53(192.168.0.101)
;; WHEN: Wed May 21 05:36:43 EDT 2014
;; MSG SIZE  rcvd: 12
```

4. While the DNS communication now works, the return code in the DNS response indicates that there may be another misconfiguration issue present. Verify the **unbound** configuration on **server1**.

- Since **REFUSED** status indicates a policy restriction, verify that the proper access policy is in place in **/etc/unbound/unbound.conf**. You should see that no access control has been granted for the subnet.

```
[student@server1 -]$ sudo grep ^[[:space:]]*access-control /etc/unbound/
unbound.conf
```

5. Fix the issue discovered and verify that it resolved the name resolution issue.

5.1. Grant the **192.168.0.0/24** subnet access to the caching nameserver in **/etc/unbound/unbound.conf**.

```
[student@server1 -]$ sudo grep ^[[:space:]]*access-control /etc/unbound/
unbound.conf
 access-control: 192.168.0.0/24 allow
```

5.2. Check the configuration for syntax errors.

```
[student@server1 -]$  sudo unbound-checkconf
unbound-checkconf: no errors in /etc/unbound/unbound.conf
```

5.3. Restart the service for the changes to take place.

```
[student@server1 -]$ sudo systemctl restart unbound
```

5.4. On **server1**, rerun the previously failed query. It should now succeed.

```
[student@server1 -]$ dig @192.168.0.101 A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @192.168.0.101 A example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25229
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.com.   IN A

;; ANSWER SECTION:
example.com.  86339 IN A 192.168.0.254

;; Query time: 0 msec
;; SERVER: 192.168.0.101#53(192.168.0.101)
;; WHEN: Wed May 21 05:56:16 EDT 2014
;; MSG SIZE  rcvd: 56
```

5.5. On **desktop1**, rerun the query to **server1**. It should now succeed.

```
[student@desktop1 -]$ dig @server1.example.com A example.com
; <<>> DiG 9.9.4-RedHat-9.9.4-14.el7 <<>> @server1.example.com A example.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25229
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.com.   IN A

;; ANSWER SECTION:
example.com.  86339 IN A 192.168.0.254

;; Query time: 0 msec
;; SERVER: 192.168.0.101#53(192.168.0.101)
;; WHEN: Wed May 21 05:56:16 EDT 2014
;; MSG SIZE  rcvd: 56
```

Summary

DNS Concepts

In this section, students learned how to:

- Review the structure of the Domain Name System.
- Detail the anatomy of DNS lookups.
- Identify common DNS resource records and their uses.

Configuring a Caching Nameserver

In this section, students learned how to:

- Identify the need for DNSSEC validation on a caching nameserver.
- Configure **unbound** DNS server as a caching nameserver.
- Administer **unbound**'s cache data.

DNS Troubleshooting

In this section, students learned how to:

- Identify symptoms and causes of common DNS issues.
- Interpret results of diagnostic tools used in DNS troubleshooting.
- Define common DNS response codes.

CHAPTER 5

CONFIGURING EMAIL TRANSMISSION

Overview	
Goal	To relay all email sent by the system through an SMTP gateway.
Objectives	• Configure a Red Hat Enterprise Linux server to transmit all email through an unauthenticated SMTP gateway.
Sections	• Configuring a Send-only Email Configuration (and Practice)
Lab	• Configuring Email Transmission

Configuring Send-only Email Service

Objectives

After completing this section, students should be able to:

Configure a Red Hat Enterprise Linux server to transmit all email through an unauthenticated SMTP gateway.

Email architecture and null clients

In today's corporate environments, email is a common method of communication. End users may use dedicated *mail clients* such as Evolution and **`mutt`** to read and send email, or the organization may have a web-based interface for its mail service.

However, Linux servers also send email, usually for automatic purposes or to report errors to an administrator. They generally use, directly or indirectly, a standard program called **`/usr/sbin/sendmail`** (provided in RHEL 7 by Postfix) to send these messages.

In practice, most servers are monitored and send out mails when incidents occur. This often requires a configured **`/usr/sbin/sendmail`** to send emails to notify the responsible system administrators by using the corporate SMTP server to transmit the messages.

A *null client* is a client machine that runs a local mail server which forwards all emails to an outbound mail relay for delivery. A null client does not accept local delivery for *any* messages, it can only send them to the outbound mail relay. Users may run mail clients on the null client to read and send emails.

This section will look at how to configure a RHEL 7 server as a Postfix null client, which will use **`sendmail`** and the SMTP protocol to transmit mail messages to the outside world through an existing outgoing mail server.

Transmission of an email message

To send an email, in most cases the mail client communicates with an outgoing mail server, which will help *relay* that message to its final destination. The mail client transmits messages to the mail server using the Simple Mail Transfer Protocol (SMTP).

The outgoing mail relay may require no authentication from internal clients, in which case the server listens on port 25/TCP. In that case, the relay will restrict which hosts can relay through IP address based restrictions or firewall rules.

Note

In cases where the outbound SMTP relay is reachable from the Internet, it is normally configured as a *mail submission agent* (MSA) for security and anti-spam reasons. An MSA listens on port 587/TCP and requires authentication of the user's mail client before accepting mail. This may be by username and password (as some webmail services provide) or through other means.

This course will only cover how to configure a basic unauthenticated null client.

The outgoing mail relay then uses DNS to look up the **MX** record identifying the mail server that accepts delivery for messages sent to the recipient's domain. The relay then uses SMTP on port 25/TCP to transmit the email to that server.

The recipient's mail service may provide a POP3 or IMAP server, such as Dovecot or Cyrus, to allow a dedicated mail client to download their messages. Frequently, the mail service provides a web-based interface, allowing clients to use a web browser as a mail client.

The following image illustrates how an email client retrieves incoming mail from an IMAP server and sends outgoing mail through an SMTP server. The mail client on **server1.example.com** fetches incoming mails from the IMAP server **imap1.example.com**. Outgoing mails are sent to **smtp1.example.com**. An **MX** DNS record defines **smtp1.example.com** as the responsible mail server for the **desktop1.example.com** domain.

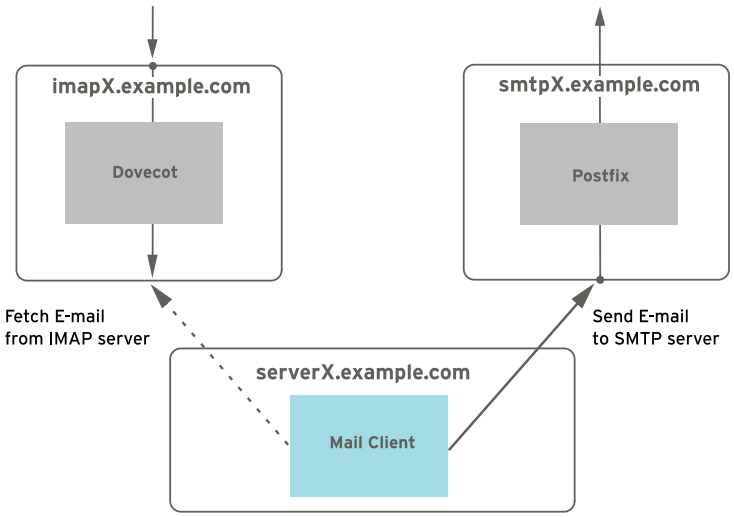

Figure 5.1: Email client communication

Postfix

Postfix is a powerful but easy-to-configure mail server. It is the default mail server in Red Hat Enterprise Linux 7. Postfix is provided by the *postfix* RPM package. It is a modular program made up of several cooperating programs. Its components are controlled by the **master** process.

The main configuration file of the **postfix** mail server is **/etc/postfix/main.cf**.

Note

There are other configuration files present in the **/etc/postfix** directory. One of the important files is **/etc/postfix/master.cf**, which controls what subservices are started.

Important Postfix Configuration Settings

Setting	Purpose
inet_interfaces	Controls which network interfaces Postfix listens on for incoming and outgoing messages. If set to **loopback-only**, Postfix listens only on **127.0.0.1** and **::1**. If set to **all**, Postfix listens on all network interfaces. One or more host names and IP addresses, separated by white space, can be listed. Default: **inet_interfaces = localhost**
myorigin	Rewrite locally posted email to appear to come from this domain. This helps ensure responses return to the correct domain the mail server is responsible for. Default: **myorigin = $myhostname**
relayhost	Forward all messages to the mail server specified that are supposed to be sent to foreign mail addresses. Square brackets around the host name suppress the MX record lookup. Default: **relayhost =**
mydestination	Configure which domains the mail server is an end point for. Email addressed to these domains are delivered into local mailboxes. Default: **mydestination = $myhostname, localhost. $mydomain, localhost**
local_transport	Determine how email addressed to **$mydestination** should be delivered. By default, set to **local:$myhostname**, which uses the **local** mail delivery agent to deliver incoming mail to the local message store in **/var/spool/mail**. Set this to **error: *error message***, e.g., **local_transport = error: local delivery disabled**, to disable local delivery completely. Default: **local_transport = local:$myhostname**
mynetworks	Allow relay through this mail server from a comma-separated list of IP addresses and networks in CIDR notation to anywhere, without further authentication. If the **mynetworks** setting is not explicitly set in **/etc/postfix/main.cf**, it will be filled automatically using the setting for **mynetworks_style**. The default for **mynetworks_style** is **subnet**, meaning that all subnets in which the server has an IP address will be added to **mynetworks**. This is often not a desired situation, especially in situations where the server has an external IP address. It is recommended that a **mynetworks** setting gets added manually, or **mynetworks_style** is set to **host**. Default: **mynetworks = 127.0.0.0/8 [::1]/128**

The configuration file **/etc/postfix/main.cf** can be edited in two ways: by hand using a text editor such as **vim**, or it can be edited using the **postconf** utility. The **postconf** command

allows for querying by individual or all settings, modifying settings, querying defaults, or showing all settings that differ from the built-in defaults:

- Run the **postconf** command without any parameter to query all settings from the **/etc/postfix/main.cf** configuration file:

```
[root@server1 ~]# postconf
2bounce_notice_recipient = postmaster
access_map_defer_code = 450
access_map_reject_code = 554
address_verify_cache_cleanup_interval = 12h
address_verify_default_transport = $default_transport
...
```

- Query a particular set of options by listing them after the **postconf** command, separated by white space. Run the following to list the **inet_interfaces** and **myorigin** options with their corresponding values:

```
[root@server1 ~]# postconf inet_interfaces myorigin
inet_interfaces = loopback-only
myorigin = $myhostname
```

Note

If a value in **/etc/postfix/main.cf** starts with a dollar sign ($), it is not a literal value, but instead points to the value of a different setting. In the previous example, the **myorigin** setting will have the same value as the **myhostname** setting. Using this syntax can simplify maintenance, since the value only has to be updated in one place.

- Run the following to add new or change existing options in the **/etc/postfix/main.cf** configuration file: **postconf -e '*setting = value*'** If there was already a setting by that name in the configuration file, it will be updated to the new value; otherwise, it will be added to the bottom of the configuration file.

  Run the following to change the **myorigin** setting to rewrite the domain part of the **FROM:** E-mail address to **example.com**:

```
[root@server1 ~]# postconf -e 'myorigin = example.com'
```

Important

The **postfix** service requires a **reload** or **restart** after the changes have been made to **/etc/postfix/main.cf**.

Note

When troubleshooting email, a log of all mail-related operations is kept in the **`systemd`** journal and **`/var/log/maillog`**, which includes information of any mail server-related actions.

The **`postqueue -p`** command displays a list of any outgoing mail messages that have been queued. To attempt to deliver all queued messages again immediately, run the **`postqueue -f`** command; otherwise, Postfix will attempt to resend them about once an hour until they are accepted or expire.

Postfix null client configuration

Remember, to act as a null client, Postfix and the RHEL system must be configured so that the following things are true:

- The **`sendmail`** command and programs that use it forward all emails to an existing outbound mail relay for delivery
- The local Postfix service does not accept local delivery for *any* email messages
- Users may run mail clients on the null client to read and send emails.

The following diagram illustrates how a null client setup works. For sending mails, the null client on **`server1`** delivers all messages to the corporate SMTP mail server **`smtp1.example.com`**.

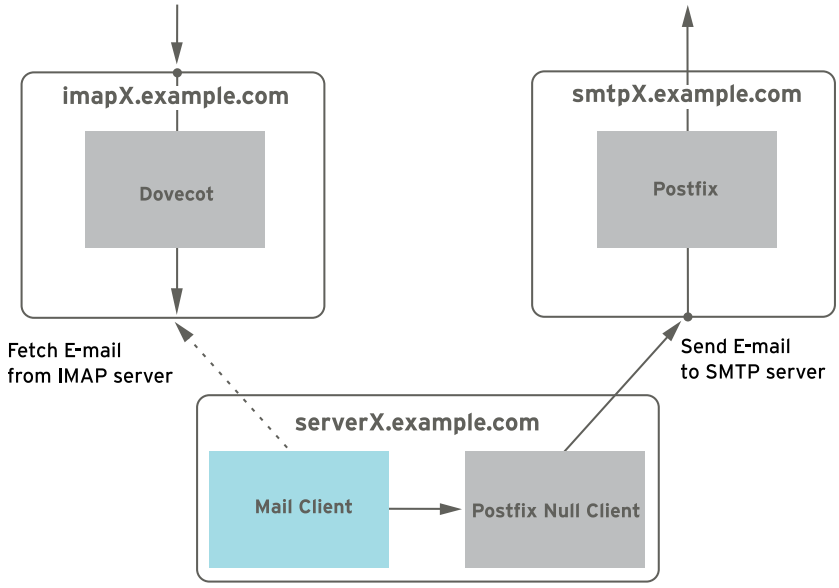

Figure 5.2: Null client communication

Note

A complete overview of all settings that are adjustable in **/etc/postfix/main.cf** and their explanations can be found in the **postconf**(5) man page. To access this man page, use the **man 5 postconf** command. Omitting the **5** will display the man page for the **postconf** command rather than the configuration guide.

Configure Postfix as null client

In this example, please follow along with these steps while your instructor demonstrates how to configure Postfix on **server1.example.com** as a null client that uses the **smtp1.example.com** mail server as a relay host. All mails passed on by the local null client have their sender address domain rewritten to **desktop1.example.com**.

Adjust the configuration of the Postfix mail server on your server1 system to act as a null client that forwards all messages to the corporate mail server on **smtp1.example.com**, which is responsible for the desktop1.example.com domain.

1. Adjust the **relayhost** directive to point to the corporate mail server. The host name of the corporate mail server needs to be enclosed in square brackets to prevent an MX record lookup with the DNS server.

```
[root@server1 ~]# postconf -e "relayhost=[smtp1.example.com]"
```

2. Configure the Postfix mail server to only relay emails from the local system.

   2.1. Let the Postfix mail server listen only on the loopback interface for emails to deliver.

```
[root@server1 ~]# postconf -e "inet_interfaces=loopback-only"
```

   2.2. Change the configuration of the null client so that mails originating from the **127.0.0.0/8** IPv4 network and the **[::1]/128** IPv6 network are forwarded to the relay host by the local null client.

```
[root@server1 ~]# postconf -e "mynetworks=127.0.0.0/8 [::1]/128"
```

3. Configure Postfix so all outgoing mails have their sender domain rewritten to the company domain **desktop1.example.com**.

```
[root@server1 ~]# postconf -e "myorigin=desktop1.example.com"
```

4. Prohibit the Postfix mail server from delivering any messages to local accounts.

   4.1. Configure the null client to not act as an end point for any mail domain. Mails where the recipient is a local email account are not accepted for local delivery. The **mydestination** option needs to be set to an empty value to achieve this.

```
[root@server1 ~]# postconf -e "mydestination="
```

4.2. Configure the local null client to not sort any mails into mailboxes on the local system. Local email delivery is turned off.

```
[root@server1 ~]# postconf -e "local_transport=error: local delivery disabled"
```

5. Restart the local **postfix** null client.

```
[root@server1 ~]# systemctl restart postfix
```

The following table shows a summary of the **/etc/postfix/main.cf** configuration file settings to configure Postfix on **server1.example.com** as a null client that uses the **smtp1.example.com** mail server as a relay host. All messages that are passed on by the local null client have their sender address domain rewritten to **desktop1.example.com**.

Null Client Postfix Settings

Directive	*Null Client* (server1.example.com)
inet_interfaces	inet_interfaces = loopback-only
myorigin	myorigin = desktop1.example.com
relayhost	relayhost = [smtp1.example.com]
mydestination	mydestination =
local_transport	local_transport = error: local delivery disabled
mynetworks	mynetworks = 127.0.0.0/8, [::1]/128

References

postconf(1), **postconf**(5), **mail**(1), and **mutt**(1) man pages

Practice: Configuring Receive and Send-only Email Service

Guided exercise

In this lab, you will configure a server to receive and relay mail and a local mail server as a null client.

Resources:	
Machines:	desktop1 and server1

Outcome:
Configure a server to relay and receive mail as well as a local mail server acting as a null client that forwards all messages to a central server for delivery.

Before you begin...

- Reset the server1 system.
- Become **root** on your **server1** system.

```
[student@server1 ~]$ su
```

- Become **root** on your **desktop1** system.

```
[student@desktop1 ~]$ sudo -i
```

1. Configure server1 to be a mail relay and to receive mail for the domain **example.com**.

   1.1. Configure postfix to listen on all interfaces.

   ```
   [root@server1 ~]# postconf -e "inet_interfaces=all"
   ```

   1.2. Configure postfix to accept mail for example.com.

   ```
   [root@server1 ~]# postconf -e "mydestination=example.com"
   ```

   1.3. Apply changes to postfix.

   ```
   [root@server1 ~]# systemctl restart postfix
   ```

   1.4. Configure and reload the firewall.

   ```
   [root@server1 ~]# firewall-cmd --permanent --add-service=smtp
   [root@server1 ~]# firewall-cmd --reload
   ```

2. Adjust the configuration of the Postfix mail server on your desktop1 system to act as a null client that forwards all mail to the corporate mail server on **server1.example.com**.

   2.1. Switch to user **root** on desktop1 and point the **relayhost** directive to server1. The host name of the relay server needs to be enclosed in square brackets to prevent an MX record lookup with the DNS server.

   ```
   [root@desktop1 ~]# postconf -e "relayhost=[server1.example.com]"
   ```

   2.2. Configure the Postfix mail server to only relay mail from the local system.

      2.2.1. Let the Postfix mail server only listen for emails to deliver on the loopback interface. Add the **inet_interfaces=loopback-only** directive to the **/etc/postfix/main.cf** configuration file.

      ```
      [root@desktop1 ~]# postconf -e "inet_interfaces=loopback-only"
      ```

      2.2.2. Change the Postfix configuration so that only messages that originate from the **127.0.0.0/8** IPv4 network and the **[::1]/128** IPv6 network are forwarded to the relay host by the null client on desktop1.

      ```
      [root@desktop1 ~]# postconf -e "mynetworks=127.0.0.0/8 [::1]/128"
      ```

   2.3. Configure the null client so that all outgoing messages have their sender domain rewritten to the company domain **example.com**.

   ```
   [root@desktop1 ~]# postconf -e "myorigin=example.com"
   ```

   2.4. Prohibit the Postfix mail server from delivering any mail to local accounts.

      2.4.1. Configure the null client to forward all mail to the relay server. The **mydestination** option needs to be set to an empty value to achieve this.

      ```
      [root@desktop1 ~]# postconf -e "mydestination="
      ```

      2.4.2. Prevent the local null client from sorting any mail into mailboxes on the desktop1 system.

      ```
      [root@desktop1 ~]# postconf -e "local_transport=error: local delivery
       disabled"
      ```

   2.5. Restart the local **postfix** null client on desktop1.

   ```
   [root@desktop1 ~]# systemctl restart postfix
   ```

   2.6. Open a new terminal on desktop1 and test the null client configuration by sending an email with **Subject: desktop1 null client** and content **null client test** to **student@example.com** with the **mail** command on desktop1. The mail command uses **/usr/sbin/sendmail**, provided by Postfix to transfer email.

```
[student@desktop1 ~]$ mail -s "desktop1 null client" student@example.com
null client test
.
EOT
```

2.7. Verify the mail arrived at the specified recipient by using the **mail** command-line mail client to **server1.example.com**.

2.7.1 Connect to **server1.example.com** as the user student and run the **mail** command-line mail client.

```
[student@server1 ~]$ mail
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/student": 1 message 1 new
>N  1 root                  Wed Dec  3 13:33  21/799   "desktop1 null
 client"
```

2.7.2 Type 1 then press ENTER to see the message

```
Message  1:
From root@example.com  Wed Dec  3 13:33:12 2014
Return-Path: <root@example.com>
X-Original-To: student@example.com
Delivered-To: student@example.com
Date: Wed, 03 Dec 2014 13:33:11 -0500
To: student@example.com
Subject: desktop1 null client
User-Agent: Heirloom mailx 12.5 7/5/10
Content-Type: text/plain; charset=us-ascii
From: root@example.com (root)
Status: R

null client test
```

Summary

Configuring Send-only Email Service

In this section, students learned how to configure the Postfix mail server to act as a null client that forwards local emails to a central SMTP server.

CHAPTER 6

PROVIDING REMOTE BLOCK STORAGE

Overview	
Goal	To provide and use networked iSCSI block devices as remote disks.
Objectives	• Explain at a high level how iSCSI is used to provide remote access to block devices. • Provide remote access using a local disk as a LUN of an iSCSI storage target. • Access remote storage using an iSCSI initiator and prepare it for use.
Sections	• iSCSI Concepts (and Practice) • Providing iSCSI Targets (and Practice) • Accessing iSCSI Storage (and Practice)
Lab	• Providing Block-based Storage

iSCSI Concepts

Objectives

After completing this section, students should be able to:

- Describe the network stack of iSCSI protocol components.
- Describe the topology distinctions between Fibre Channel and iSCSI.
- Describe the distinction between block I/O and file I/O.

Introduction to iSCSI

The Internet Small Computer System Interface (iSCSI) is a TCP/IP-based protocol for emulating a SCSI high-performance local storage bus over IP networks, providing data transfer and management to remote block storage devices. As a storage area network (SAN) protocol, iSCSI extends SANs across local and wide area networks (LANs, WANs, and the Internet), providing location-independent data storage retrieval with distributed servers and arrays.

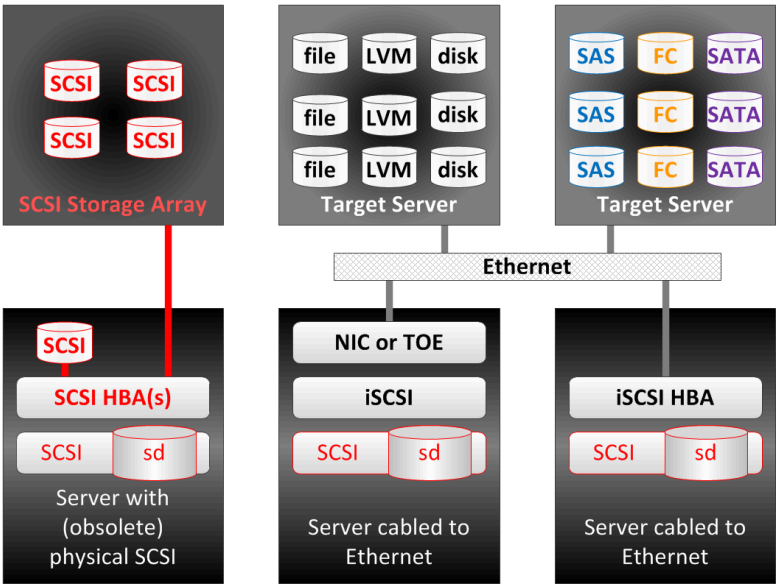

Figure 6.1: SCSI and iSCSI block storage topologies

The SCSI protocol suite provides the Command Descriptor Block (CDB) command set over a device bus communication protocol. The original SCSI topology used physical cabling with a 20-meter limitation for all devices per channel (cabled bus). Devices used unique numeric target IDs (0-7, or 0-15 with dual channel). Physical SCSI disks and cabling were obsoleted by popular implementation of Fibre Channel (FC), which retained the SCSI CDB command set but replaced the disk and bus communication with protocols for longer and faster optical cabling.

The iSCSI protocol also retains the CDB command set, performing bus communication between iSCSI systems that is encapsulated over standard TCP/IP. iSCSI servers emulate SCSI devices using files, logical volumes, or disks of any type as the underlying storage (*backstore*) presented as *targets*. An iSCSI service is typically implemented in software above either an operating system TCP/IP stack or a TCP offload engine (TOE), a specialized Ethernet network interface card (NIC)

that includes the TCP/IP network layers to increase performance. iSCSI can also be hardware-implemented as a host bus adapter (HBA) for a greater performance increase.

Enterprise-grade SANs require dedicated traffic infrastructure. FC's independent optical cabling and switches guarantees isolation. iSCSI should also be implemented on cabling that is independent of standard LAN traffic, since performance can degrade due to bandwidth congestion on shared networks. Both Ethernet and FC now offer copper and optical cabling options, allowing network consolidation combined with traffic classification.

Storage area network traffic is typically unencrypted, since physical server-to-storage cabling is normally enclosed within secure data centers. For WAN security, iSCSI and Fibre Channel over Ethernet (FCoE) can utilize Internet Protocol Security (IPSec), a protocol suite for securing IP network traffic. Select networking hardware (preferred NICs, TOEs, and HBAs) can provide encryption. iSCSI offers Challenge-Handshake Authentication Protocol (CHAP) usernames and passwords as an authentication mechanism to limit connectivity between chosen initiators and targets.

Until recently, iSCSI was not considered an enterprise-grade storage option, primarily due to the use of slower 100 and 1000 Mb/s Ethernet, compared to FC's 1 and 4 Gb/s optical infrastructure. With current 10 or 40 Gb/s Ethernet and 8, 10, 16, or 20 Gb/s FC, and pending 100 Gb/s Ethernet and 32 or 128 Gb/s FC, bandwidth availability is now similar for both.

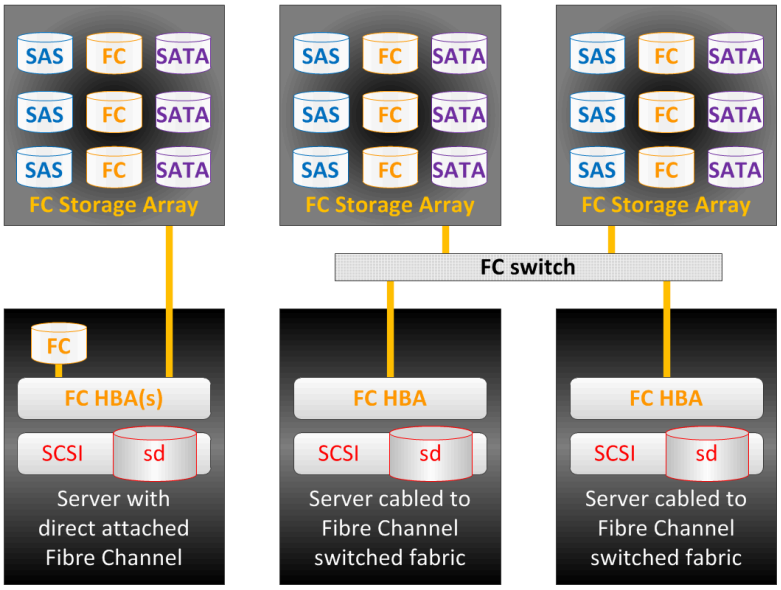

Figure 6.2: Fibre Channel block storage topologies

The use of iSCSI extends a SAN beyond the limits of local cabling, facilitating storage consolidation in local or remote data centers. Because iSCSI structures are logical, new storage allocations are made using only software configuration, without the need for additional cable or physical disks. iSCSI also simplifies data replication, migration and disaster recovery using multiple remote data centers.

iSCSI fundamentals

The iSCSI protocol functions in a familiar client-server configuration. Client systems configure *initiator* software to send SCSI commands to remote server storage *targets*. Accessed iSCSI targets appear on the client system as local, unformatted SCSI block devices, identical to devices connected with SCSI cabling, FC direct attached, or FC switched fabric.

iSCSI Component Terminology

Term	Description
initiator	An iSCSI client, typically available as software but also implemented as iSCSI HBAs. Initiators must be given unique names (see **IQN**).
target	An iSCSI storage resource, configured for connection from an iSCSI server. Targets must be given unique names (see **IQN**). A target provides one or more numbered block devices called logical units (see **LUN**). An iSCSI server can provide many targets concurrently.
ACL	An Access Control List (entry), an access restriction using the node IQN (commonly the iSCSI Initiator Name) to validate access permissions for an initiator.
discovery	Querying a target server to list configured targets. Target use requires an additional access steps (see **login**).
IQN	An iSCSI Qualified Name, a worldwide unique name used to identify both initiators and targets, in the mandated naming format: **iqn.YYYY-MM.com.reversed.domain[:optional_string]** **iqn**—Signifying that this name will use a domain as its identifier. **YYYY-MM**—The first month in which the domain name was owned. **com.reversed.domain**—The reversed domain name of the organization creating this iSCSI name. **optional_string**—An optional, colon-prefixed string assigned by the domain owner as desired while remaining worldwide unique. It may include colons to separate organization boundaries.
login	Authenticating to a target or LUN to begin client block device use.
LUN	A Logical Unit Number, numbered block devices attached to and available through a target. One or more LUNs may be attached to a single target, although typically a target provides only one LUN.
node	Any iSCSI initiator or iSCSI target, identified by its IQN.
portal	An IP address and port on a target or initiator used to establish connections. Some iSCSI implementations use *portal* and *node* interchangeably.
TPG	Target Portal Group, the set of interface IP addresses and TCP ports to which a specific iSCSI target will listen. Target configuration (e.g., ACLs) can be added to the TPG to coordinate settings for multiple LUNs.

iSCSI uses ACLs to perform *LUN masking*, managing the accessibility of appropriate targets and LUNs to initiators. Access to targets may also be limited with CHAP authentication. iSCSI ACLs are similar to FC's use of device worldwide numbers (WWNs) for *soft zoning* management restrictions. Although FC switch-level compulsory port restriction (*hard zoning*) has no comparable iSCSI mechanism, Ethernet VLANs could provide similar isolation security.

Unlike local block devices, iSCSI network-accessed block devices are discoverable from many remote initiators. Typical local file systems (e.g., ext4, XFS, btrfs) do not support concurrent multisystem mounting, which can result in significant file system corruption. Clustered systems resolve multiple system access by use of the Global File System (GFS2), designed to provide distributed file locking and concurrent multinode file system mounting.

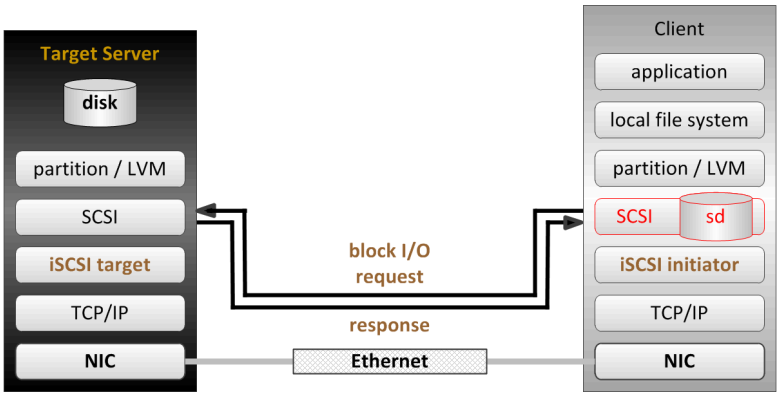

Figure 6.3: Block I/O network stack components

An attached iSCSI block device appears as a local SCSI block device (**sdx**) for use underneath a local file system, swap space, or a raw database installation as diagrammed previously. See the following for contrast with the use of network file server protocols (e.g., NFS, SMB), providing file I/O from remote file systems to local applications on multiple client systems concurrently.

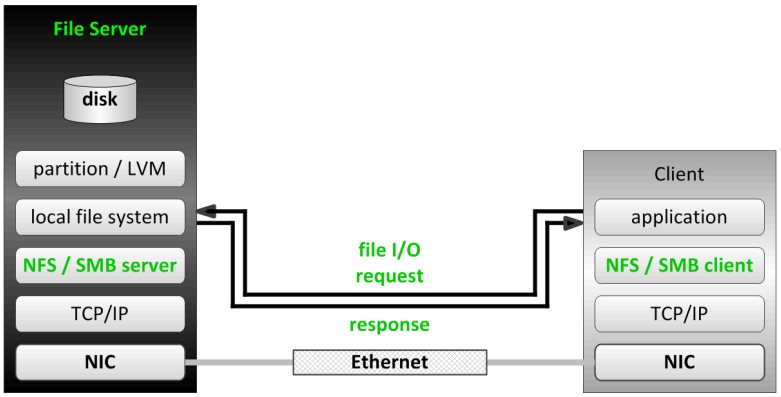

Figure 6.4: File I/O network stack components

References

Open-iSCSI project
http://www.open-iscsi.org/

Technical Committee T10 - SCSI Storage Interfaces
http://www.t10.org/

RFC 3270 - Internet Small Computer Systems Interface (iSCSI)
http://www.ietf.org/rfc/rfc3720.txt

Providing iSCSI Targets

Objectives

After completing this section, students should be able to:

- Prepare a Red Hat Enterprise Linux server to provide the iSCSI target service.
- Build iSCSI targets and LUNs for network access.
- Limit access to iSCSI targets and LUNs.

iSCSI target overview

In original SCSI protocol terminology, a target is a single connectible storage or output device uniquely identified on a SCSI bus. In iSCSI, in which the SCSI bus is emulated across an IP network, a target can be a dedicated physical device in a network-attached storage enclosure or an iSCSI software-configured logical device on a networked storage server. A target, like HBAs and initiators, is an end point in SCSI bus communication, passing command descriptor blocks (CDB) to request or provide storage transactions.

To provide access to the storage or output device, a target is configured with one or more logical unit numbers (LUNs). In iSCSI, LUNs appear as the target's sequentially numbered disk drives, although targets typically have only one LUN. An initiator performs SCSI negotiation with a target to establish a connection to the LUN. The LUN responds as an emulated SCSI disk block device, which can be used in raw form or formatted with a client-supported file system.

Warning

Do not mount single-system file systems to more than one system at a time. iSCSI allows shared target and LUN access from multiple initiator nodes, requiring the use of cluster-capable file systems such as GFS2. Mounting standard file systems designed for local, single-system access (e.g., btrfs, ext3, ext4, FAT32, HPFS+, NTFS, XFS, ZFS) from more than one system concurrently will cause file system corruption.

iSCSI provides for *LUN masking* by using ACLs to restrict LUN accessibility to specific initiators. Except when shared access is intended, ACLs can ensure that only a designated client node can login to a specific target. On the target server, ACLs can be set at the TPG level to secure groups of LUNs, or set individually per LUN.

iSCSI target configuration

Target server configuration demonstration

`targetcli` is both a command-line utility and an interactive shell in which to create, delete, and configure iSCSI target components. Target stack objects are grouped into a hierarchical tree of objects, allowing easy navigation and contextual configuration. Familiar Linux commands are used in this shell: **`cd`**, **`ls`**, **`pwd`**, and **`set`**.

The **`targetcli`** shell also supports **`TAB`** completion. Administrators can use **`TAB`** to either complete partially typed commands or view a list of acceptable key-words at the current location in a command.

In this example, please follow along with the steps below while your instructor demonstrates using **targetcli** to configure a target server.

1. Install **targetcli** if needed.

```
[root@server0 ~]# yum -y install targetcli
```

2. Run **targetcli** with no options to enter interactive mode.

```
[root@server0 ~]# targetcli
/> ls
```

```
o- / ..................................................................... [...]
  o- backstores ........................................................... [...]
  | o- block ............................................... [Storage Objects: 0]
  | o- fileio .............................................. [Storage Objects: 0]
  | o- pscsi ............................................... [Storage Objects: 0]
  | o- ramdisk ............................................. [Storage Objects: 0]
  o- iscsi ......................................................... [Targets: 0]
  o- loopback ...................................................... [Targets: 0]
```

Figure 6.5: Initial targetcli blank configuration

3. Create backing storage (backstores). There are several types of backing storage.
   - **block** - A block device defined on the server. A disk drive, disk partition, a logical volume, multipath device, any device files defined on the server that are of type **b**.
   - **fileio** - Create a file, of a specified size, in the filesystem of the server. This method is similar to using image files to be the storage for virtual machine disk images.
   - **pscsi** - Physical SCSI. Permits passthrough to a physical SCSI device connected to the server. This backstore type is not typically used.
   - **ramdisk** - Create a ramdisk device, of a specified size, in memory on the server. This type of storage will not store data persistently. When the server is rebooted, the ramdisk definition will return when the target is instantiated, but all data will have been lost.

   Examples include using an existing logical volume, a disk partition, and a new file at a specified size. The backstores display as **deactivated**.

```
/> cd /backstores/
/backstores> block/ create block1 /dev/iSCSI_vg/disk1_lv
Created block storage object block1 using /dev/iSCSI_vg/disk1_lv.
/backstores> block/ create block2 /dev/vdb2
Created block storage object block2 using /dev/vdb2.
/backstores> fileio/ create file1 /root/disk1_file 100M
Created fileio file1 with size 104857600
/backstores> ls
o- backstores ..................................................... [...]
  o- block ......................................... [Storage Objects: 2]
  | o- block1 ....... [/dev/iSCSI_vg/disk1_lv (100.0MiB) write-thru deactivated]
  | o- block2 ..................... [/dev/vdb2 (1.0GiB) write-thru deactivated]
  o- fileio ........................................ [Storage Objects: 1]
  | o- file1 ............. [/root/disk1_file (100.0MiB) write-back deactivated]
  o- pscsi ......................................... [Storage Objects: 0]
  o- ramdisk ....................................... [Storage Objects: 0]
```

4. Create an IQN for the target. This step will also create a default TPG underneath the IQN.

```
/backstores> cd /iscsi/
/iscsi> create iqn.2014-06.com.example:remotedisk1
Created target iqn.2014-06.com.example:remotedisk1.
Created TPG 1.
/iscsi> ls
o- iscsi ..................................................... [Targets: 1]
  o- iqn.2014-06.com.example:remotedisk1 ....................... [TPGs: 1]
    o- tpg1 ....................................... [no-gen-acls, no-auth]
      o- acls ..................................................... [ACLs: 0]
      o- luns ..................................................... [LUNs: 0]
      o- portals ................................................. [Portals: 0]
```

An administrator can use the **`create`** without specifying the IQN to create. **`targetcli`** will generate an IQN similar to the following: **`iqn.2003-01.org.linux-iscsi.server0.x8664:sn.69b30d2cfd01`**. Specifying the IQN value provides the ability for an administrator to use a meaningful namespace for their IQNs.

5. In the TPG, create an ACL for the client node to be used later. Because the global **`auto_add_mapped_luns`** parameter is set to true (default), any existing LUNs in the TPG are mapped to each ACL as it is created.

```
/iscsi> cd iqn.2014-06.com.example:remotedisk1/tpg1/
/iscsi/iqn.20...sk1/tpg1> acls/ create iqn.2014-06.com.example:desktop0
Created NODE ACL for iqn.2014-06.com.example:desktop0
/iscsi/iqn.20...:server0/tgp1> ls
o- tpg1 ....................................... [no-gen-acls, no-auth]
  o- acls ..................................................... [ACLs: 1]
  | o- iqn.2014-06.com.example:desktop0 ..................... [Mapped LUNs: 0]
  o- luns ..................................................... [LUNs: 0]
  o- portals ................................................. [Portals: 0]
```

This ACL configures the target to only accept initiator connections from a client presenting **`iqn.2014-06.com.example:desktop0`** as it's initiator IQN, also known as the initiator name.

6. In this TPG, create a LUN for each existing backstores. This step also activates each backstore. Because acls exist for the TPG, they will be automatically assigned to each LUN created.

```
/iscsi/iqn.20...:server0/tgp1> luns/ create /backstores/block/block1
Created LUN 0.
Created LUN 0->0 mapping in node ACL iqn.2014-06.com.example:desktop0
/iscsi/iqn.20...:server0/tgp1> luns/ create /backstores/block/block2
Created LUN 1.
Created LUN 1->1 mapping in node ACL iqn.2014-06.com.example:desktop0
/iscsi/iqn.20...:server0/tgp1> luns/ create /backstores/fileio/file1
Created LUN 2.
Created LUN 2->2 mapping in node ACL iqn.2014-06.com.example:desktop0
/iscsi/iqn.20...:server0/tgp1> ls
o- tpg1 ....................................... [no-gen-acls, no-auth]
  o- acls ..................................................... [ACLs: 1]
  | o- iqn.2014-06.com.example:desktop0 ..................... [Mapped LUNs: 3]
  |   o- mapped_lun0 ............................... [lun0 block/block1 (rw)]
  |   o- mapped_lun1 ............................... [lun1 block/block2 (rw)]
  |   o- mapped_lun2 ............................... [lun2 fileio/file1 (rw)]
```

```
  o- luns ..................................................... [LUNs: 3]
  | o- lun0 ....................... [block/block1 (/dev/iSCSI_vg/disk1_lv)]
  | o- lun1 ..................................... [block/block2 (/dev/vdb2)]
  | o- lun2 ............................... [fileio/file1 (/root/disk1_file)]
  o- portals ................................................... [Portals: 0]
```

Having three LUNs assigned to a target means that when the initiator connects to the target, it will receive three new SCSI devices.

7. Still inside the TPG, create a portal configuration to designate the listening IP address and ports. Create a portal using the system's public network interface. Without specifying a TCP port to use, the portal creation will default to the standard iSCSI port (3260).

```
/iscsi/iqn.20...:server0/tgp1> portals/ create 192.168.0.101
Using default IP port 3260
Created network portal 192.168.0.101:3260
/iscsi/iqn.20...:server0/tgp1> ls
o- tpg1 ........................................... [no-gen-acls, no-auth]
  o- acls ..................................................... [ACLs: 1]
  | o- iqn.2014-06.com.example:desktop0 ..................... [Mapped LUNs: 3]
  |   o- mapped_lun0 ............................... [lun0 block/block1 (rw)]
  |   o- mapped_lun1 ............................... [lun1 block/block2 (rw)]
  |   o- mapped_lun2 ............................... [lun2 fileio/file1 (rw)]
  o- luns ..................................................... [LUNs: 3]
  | o- lun0 ....................... [block/block1 (/dev/iSCSI_vg/disk1_lv)]
  | o- lun1 ..................................... [block/block2 (/dev/vdb2)]
  | o- lun2 ............................... [fileio/file1 (/root/disk1_file)]
  o- portals ................................................... [Portals: 1]
    o- 192.168.0.101:3260 ..................................... [OK]
```

If the IP is not specified with the portal creation, an IP of 0.0.0.0 will be used. This will permit connections on all network interfaces defined on the server.

8. View the entire configuration, then exit **targetcli**. **targetcli** will automatically save upon exit. The resulting persistent configuration file is stored in JavaScript Object Notation (JSON) format.

```
/iscsi/iqn.20...:server0/tgp1> cd /
/> ls
o- / ......................................................................... [...]
  o- backstores .............................................................. [...]
  | o- block ................................... [Storage Objects: 2]
  | | o- block1 ....... [/dev/iSCSI_vg/disk1_lv (100.0MiB) write-thru activated]
  | | o- block2 ..................... [/dev/vdb2 (1.0GiB) write-thru activated]
  | o- fileio ................................... [Storage Objects: 1]
  | | o- file1 .............. [/root/disk1_file (100.0MiB) write-back activated]
  | o- pscsi .................................... [Storage Objects: 0]
  | o- ramdisk .................................. [Storage Objects: 0]
  o- iscsi ............................................... [Targets: 1]
  | o- iqn.2014-06.com.example:remotedisk1 ..................... [TPGs: 1]
  |   o- tpg1 ..................................... [no-gen-acls, no-auth]
  |     o- acls ..................................................... [ACLs: 1]
  |     | o- iqn.2014-06.com.example:desktop0 ................. [Mapped LUNs: 3]
  |     |   o- mapped_lun0 ........................ [lun0 block/block1 (rw)]
  |     |   o- mapped_lun1 ........................ [lun1 block/block2 (rw)]
  |     |   o- mapped_lun2 ........................ [lun2 fileio/file1 (rw)]
  |     o- luns ..................................................... [LUNs: 3]
  |     | o- lun0 ..................... [block/block1 (/dev/iSCSI_vg/disk1_lv)]
  |     | o- lun1 ................................. [block/block2 (/dev/vdb2)]
```

```
  |     | o- lun2 ........................... [fileio/file1 (/root/disk1_file)]
  |     o- portals ........................................... [Portals: 1]
  |       o- 192.168.0.101:3260 ............................................ [OK]
  o- loopback ..................................................... [Targets: 0]
/> exit
Global pref auto_save_on_exit=true
Last 10 configs saved in /etc/target/backup.
Configuration saved to /etc/target/saveconfig.json
```

9. Add a port exemption to the default firewall for port 3260, the standard iSCSI port.

```
[root@server0 ~]# firewall-cmd --add-port=3260/tcp
[root@server0 ~]# firewall-cmd --add-port=3260/tcp --permanent
```

10. Enable the **target.service** systemd unit. The **target.service** will recreate the target configuration from the json file at boot. If this step is skipped, any configured targets will work until the machine is rebooted; however, after a reboot, no targets will be offered by the server.

```
[root@server0 ~]# systemctl enable target
```

Authentication

In addition to ACL node verification, password-based authentication can be implemented. Authentication can be required during the iSCSI discovery phase. Authentication can be unidirectional or bidirectional.

CHAP authentication does not use strong encryption for the passing of credentials. While CHAP does offer an additional factor of authentication besides having a correctly configured initiator name, configured in an ACL, it should not be considered secure. If security of iSCSI data is a concern, controlling the network side of the protocol is a better method to assure security. Providing a dedicated, isolated network, or vlans to pass the iSCSI traffic will be a more secure implementation of the protocol.

Command-line mode

In the demonstration, **targetcli** was run in interactive mode, but **targetcli** can also be used to execute a series of commands via the command-line. In the following example, **targetcli** will be used to create a backstore device, an IQN, and activate a portal. The example, as written, will not present a usable target, but is meant to demonstrate several actions executed with **targetcli**. This method could be used to script target configuration. At the end of the list of commands, a **saveconfig** command is executed. Unlike the interactive use of **targetcli**, command-line mode will not save the configuration as a json file until the **saveconfig** command is used.

```
[root@server0 ~]# targetcli /backstores/block create block1 /dev/vdb
Created block storage object block1 using /dev/vdb.
[root@server0 ~]# targetcli /iscsi create iqn.2014-06.com.example:remotedisk1
Created target iqn.2014-06.com.example:remotedisk1.
Created TPG 1.
[root@server0 ~]# targetcli /iscsi/iqn.2014-06.com.example:remotedisk1/tpg1/portals
 create 192.168.0.101
Using default IP port 3260
Created network portal 192.168.0.101:3260.
[root@server0 ~]# targetcli saveconfig
Last 10 configs saved in /etc/target/backup.
```

```
Configuration saved to /etc/target/saveconfig.json
```

References

targetcli(8) man page

> LINUX I/O Targetcli
> http://linux-iscsi.org/wiki/Targetcli

Practice: Providing iSCSI Targets

Guided exercise

In this lab, you will configure a RHEL server to become an iSCSI target server, including opening firewall access, creating a backing store, and setting target and LUN access parameters.

Resources:	
Files:	/etc/target/saveconfig.json
Machines:	server1

Outcomes:
The creation and configuration of an iSCSI target block device with ACL-validated access.

Before you begin...
- Reset your **`server1`** system.
- Log in to your server1 system and become **`root`**.

```
[student@server1 ~]$ su
```

1. Prepare the server system to become an iSCSI target server by installing the target configuration utility, starting the **`target`** service and opening the firewall for the iSCSI server port.

   1.1. Install the *targetcli* RPM.

   ```
   [root@server1 ~]# yum -y install targetcli
   ```

   1.2. Enable and start the **`target`** service.

   ```
   [root@server1 ~]# systemctl enable target; systemctl start target
   ```

   1.3. Open the iSCSI server port on the firewall as a permanent change. Reload the configuration for immediate use.

   ```
   [root@server1 ~]# firewall-cmd --permanent --add-port=3260/tcp
   [root@server1 ~]# firewall-cmd --reload
   ```

2. Create the physical disk structure to use as a backing store device for a target to be created in a later step. Partition a disk, build a volume group, and create a logical volume.

   2.1. Create a 1 GB partition using the second physical disk on the server system.

   ```
   [root@server1 ~]# fdisk /dev/vdb
   ```

Use **fdisk** to create a new primary partition with a size of 1 GB. Use the partition tag appropriate for use as a Linux LVM partition ("**8e**"). Remember to write before exiting. This exercise uses the expected partition name **/dev/vdb1**.

2.2. Create a logical volume manager volume group named **iSCSI_vg** using the partition created in the previous step.

```
[root@server1 services]# pvcreate /dev/vdb1
[root@server1 services]# vgcreate iSCSI_vg /dev/vdb1
[root@server1 services]# vgdisplay iSCSI_vg
```

2.3. Create a 100 MiB logical volume named **disk1_lv** in the new volume group.

```
[root@server1 ~]# lvcreate -n disk1_lv -L 100m iSCSI_vg
[root@server1 ~]# lvdisplay iSCSI_vg/disk1_lv
```

3. Go into **targetcli**'s interactive mode to configure the iSCSI target.

```
[root@server1 ~]# targetcli
```

4. Configure the existing **/dev/iSCSI_vg/disk1_lv** logical volume as a block-type backing store. Use the name **server1.disk1** for the backstore, replacing 1 with your system number.

```
/> /backstores/block/ create server1.disk1 /dev/iSCSI_vg/disk1_lv
```

5. Create a unique iSCSI Qualified Name (IQN) for the target. The target will be **iqn.2014-06.com.example:server1**.

```
/> /iscsi create iqn.2014-06.com.example:server1
```

Creating the IQN name automatically created a default target portal group named **tpg1**.

6. Create an ACL for the client node (initiator). The initiator will be connecting with it's initiator name set to **iqn.2014-06.com.example:desktop1**.

```
/> /iscsi/iqn.2014-06.com.example:server1/tpg1/acls/ create
 iqn.2014-06.com.example:desktop1
```

7. Create a LUN under the target, the LUN should use the previously defined backing storage device named **server1.disk1**.

```
/> /iscsi/iqn.2014-06.com.example:server1/tpg1/luns create /backstores/block/
server1.disk1
```

8. Configure a portal for the target to listen on 192.168.0.101, port 3260.

```
/> /iscsi/iqn.2014-06.com.example:server1/tpg1/portals create 192.168.0.101
```

9. View, verify, and save the target server configuration.

   9.1. View and verify the configuration.

```
/> ls
o- / ......................................................... [...]
  o- backstores .............................................. [...]
  | o- block ................................... [Storage Objects: 1]
  | | o- server1.disk1 .....
  |           .....[/dev/iSCSI_vg/disk1_lv (100.0MiB) write-thru activated]
  | o- fileio .................................. [Storage Objects: 0]
  | o- pscsi ................................... [Storage Objects: 0]
  | o- ramdisk ................................. [Storage Objects: 0]
  o- iscsi ............................................ [Targets: 1]
  | o- iqn.2014-06.com.example:server1 ....................... [TPGs: 1]
  |   o- tpg1 ................................ [no-gen-acls, no-auth]
  |     o- acls ........................................... [ACLs: 1]
  |     | o- iqn.2014-06.com.example:desktop1 ......... [Mapped LUNs: 1]
  |     | | o- mapped_lun0 ............ [lun1 block/server1.disk1 (rw)]
  |     o- luns ........................................... [LUNs: 1]
  |     | o- lun0 ...... [block/server1.disk1 (/dev/iSCSI_vg/disk1_lv)]
  |     o- portals ..................................... [Portals: 1]
  |       o- 192.168.0.101:3260 ..................................... [OK]
  o- loopback ........................................... [Targets: 0]
```

   9.2. Exit and save the configuration to the default **`/etc/target/saveconfig.json`**.

```
/> exit
```

Important

The outcome of the practice will be used in the next practice. Therefore, do not reset your server1 host after the completion of this practice.

Note

A practical method to test this configuration is to install the iSCSI client tools package and perform a discovery from a separate client system. This is presented in the *Accessing iSCSI Storage* exercise.

Although an iSCSI client configuration can be installed on this target server, an administrator must ensure that any iSCSI node records created during test discoveries are deleted prior to target discovery from client systems. These procedures are presented in the *Accessing iSCSI Storage* section.

Accessing iSCSI Storage

Objectives

After completing this section, students should be able to:

- Prepare a Red Hat Enterprise Linux client to access an iSCSI target service.
- Access and log into a remote iSCSI portal, target, and LUN.
- Practice iSCSI persistence through the use and deletion of persistent node records.

iSCSI initiator introduction

In Red Hat Enterprise Linux an iSCSI initiator is typically implemented in software, and functions similar to a hardware iSCSI HBA to access targets from a remote storage server. Using a software-based iSCSI initiator requires connecting to an existing Ethernet network of sufficient bandwidth to carry the expected storage traffic.

iSCSI can also be implemented using a hardware initiator that includes the required protocols in a dedicated host bus adapter. iSCSI HBAs and TCP offload engines (TOE), which include the TCP network stack on an Ethernet NIC, move the processing of iSCSI or TCP overhead and Ethernet interrupts to hardware, easing the load on system CPUs.

Configuring an iSCSI client initiator requires installing the *iscsi-initiator-utils* package, which includes the **iscsi** and **iscsid** services and the **/etc/iscsi/iscsid.conf** and **/etc/iscsi/initiatorname.iscsi** configuration files.

As an iSCSI node, the client requires a unique IQN. The default **/etc/iscsi/initiatorname.iscsi** file contains a generated IQN using Red Hat's domain. Administrators typically reset the IQN to their own domain and an appropriate client system string.

The **/etc/iscsi/iscsid.conf** file contains default settings for node records created during new target discovery. Settings include iSCSI timeouts, retry parameters, and authentication usernames and passwords. Changing this file requires restarting the **iscsi** service.

```
[root@desktop1 ~]# systemctl restart iscsi
```

To be able to discover targets, install the *iscsi-initiator-utils* package, then enable and start the **iscsi** service. Targets must be discovered before device connection and use. The discovery process stores target node information and settings in the **/var/lib/iscsi/nodes** directory, using defaults from **/etc/iscsi/iscsid.conf**. Since the same target can exist on multiple portals, node records are stored for each portal. Perform discovery with the following command:

```
[root@desktop1 ~]# iscsiadm -m discovery -t sendtargets -p target_server[:port]
192.168.0.101:3260,1 iqn.2014-06.com.example:server1
```

In discovery mode, the **sendtargets** request returns only targets with access configured for this initiator. The port number can be omitted when the target server is configured on default port 3260. Upon discovery, a node record is written to **/var/lib/iscsi/nodes** and used for subsequent logins. To use the listed target, log in using the following command:

```
[root@desktop1 ~]# iscsiadm -m node -T iqn.2014-06.com.example:server1 [-
p target_server[:port]] -l
```

Specifying the portal is optional. If the target exists on multiple portals (e.g., in a multipathed, redundant server configuration), performing a login without specifying a portal will connect to every portal node that accepts this target name.

Once discovered, obtain information about targets with the **iscsiadm** command. Use the option **-P *N*** to set the command detail level, with **0** specifying the least verbose output.

- **iscsiadm -m discovery [-P 0|1]**: Show information about discovered targets.
- **iscsiadm -m node [-P 0|1]**: Show information about known targets.
- **iscsiadm -m session [-P 0|1|2|3]**: Show information about active sessions.

To discontinue using a target, use **iscsiadm** to log out temporarily. By design, node records remain after logout and are used to automatically log into targets upon system reboot or **iscsi** service restart. Log out of a target using the following command (notice the similarity to login):

```
[root@desktop1 ~]# iscsiadm -m node -T iqn.2012-04.com.example:example [-
p target_server[:port]] -u
```

If a portal is not specified, the target logs out of all relevant portals. To log into the target again, repeating discovery is not necessary since the node records already exist. Permanently logging out of a target requires deleting the node records so that manual or automatic login cannot reoccur without performing another discovery. Not specifying a portal removes the target node records for all relevant portals. Delete the node record permanently by using the following command (notice again the command similarity):

```
[root@desktop1 ~]# iscsiadm -m node -T iqn.2012-04.com.example:example [-
p target_server[:port]] -o delete
```

References

iscsiadm(8), **iscsid**(8) man pages

> Open-iSCSI
> http://www.open-iscsi.org

Practice: Accessing iSCSI Storage

Guided exercise

In this lab, you will discover targets on an iSCSI target portal, then practice manual and automatic login and logout of an iSCSI target.

Resources:	
Files:	/etc/iscsi/initiatorname.iscsi /var/lib/iscsi/nodes/*
Machines:	desktop1

Outcomes:
Discover and connect to an access-controlled remote iSCSI target and LUN.

Before you begin...
This practice uses the outcome of the previous lab. DO NOT reset server1.

From your home directory on desktop1, switch user to root.

1. Prepare the client system to become an iSCSI initiator node by installing the initiator utilities, setting the unique iSCSI client name, and starting the iSCSI client service.

   1.1. Install the *iscsi-initiator-utils* RPM, if not already installed.

   ```
   [root@desktop1 ~]# yum install -y iscsi-initiator-utils
   ```

   1.2. Create a unique iSCSI Qualified Name for the client initiator by modifying the **`InitiatorName`** setting in **`/etc/iscsi/initiatorname.iscsi`**. Use the client system name as the optional string after the colon.

   ```
   [root@desktop1 ~]# echo "InitiatorName=iqn.2014-06.com.example:desktop1" > /etc/
   iscsi/initiatorname.iscsi
   [root@desktop1 ~]# cat /etc/iscsi/initiatorname.iscsi
   InitiatorName=iqn.2014-06.com.example:desktop1
   ```

   1.3. Enable and start the **`iscsi`** client service.

   ```
   [root@desktop1 ~]# systemctl enable iscsi; systemctl start iscsi
   ```

2. Discover and log into the configured target from the iSCSI target server.

   2.1. Discover the configured iSCSI target(s) provided by the iSCSI target server portal.

   ```
   [root@desktop1 ~]# iscsiadm -m discovery -t st -p 192.168.0.101
   192.168.0.101:3260,1 iqn.2014-06.com.example:server1
   ```

   2.2. Log into the presented iSCSI target.

```
[root@desktop1 ~]# iscsiadm -m node -T iqn.2014-06.com.example:server1 -p
 192.168.0.101 -l
```

2.3. Identify the newly available block device created by the iSCSI target login.

```
[root@desktop1 ~]# dmesg|tail
...OMITTED...
[91737.871506] scsi 2:0:0:0: Attached scsi generic sg1 type 0
[91738.097103] sd 2:0:0:0: [sda] 204800 512-byte logical blocks: (104 MB/100
 MiB)
[91738.104731] sd 2:0:0:0: [sda] Write Protect is off
[91738.105960] sd 2:0:0:0: [sda] Mode Sense: 43 00 10 08
[91738.108551] sd 2:0:0:0: [sda] Write cache: enabled, read cache: enabled,
 supports DPO and FUA
[91738.137092]  sda: unknown partition table
[91738.151598] sd 2:0:0:0: [sda] Attached SCSI disk
[root@desktop1 ~]# lsblk
NAME                      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0                         2:0    1    4K  0 disk
sda                         8:0    0  100M  0 disk
...OMITTED...
[root@desktop1 ~]# tail /var/log/messages
...OMITTED...
Dec  3 13:55:31 desktop1 kernel: sda: unknown partition table
Dec  3 13:55:31 desktop1 kernel: sd 2:0:0:0: [sda] Attached SCSI disk
Dec  3 13:55:31 desktop1 iscsid: Connection1:0 to [target:
 iqn.2014-06.com.example:server1, portal: 192.168.0.101,3260] through [iface:
 default] is operational now
```

3. Browse the connection information about the target portal, connection, and parameters used by the connected device. Locate the node record.

3.1. List the targets recognized by the **iscsi** service.

```
[root@desktop1 ~]# iscsiadm -m session -P 3
iSCSI Transport Class version 2.0-870
version 6.2.0-873-21
Target: iqn.2014-06.com.example:server1
        Current Portal: 192.168.0.101:3260,1
        Persistent Portal: 192.168.0.101:3260,1
                **********
                Interface:
                **********Resources
                Iface Name: default
                Iface Transport: tcp
                Iface Initiatorname: iqn.2014-06.com.example:desktop1
                Iface IPaddress: 192.168.0.1
                Iface HWaddress: <empty>
                Iface Netdev: <empty>
                SID: 3
                iSCSI Connection State: LOGGED IN
                iSCSI Session State: LOGGED_IN
                Internal iscsid Session State: NO CHANGE
                ************************
                Negotiated iSCSI params:
                ************************
                HeaderDigest: None
                DataDigest: None
                MaxRecvDataSegmentLength: 262144
```

```
                MaxXmitDataSegmentLength: 8192
                FirstBurstLength: 65536
                MaxBurstLength: 262144
                ImmediateData: Yes
                InitialR2T: Yes
                MaxOutstandingR2T: 1
                ************************
                Attached SCSI devices:
                ************************
                Host Number: 5  State: running
                scsi5 Channel 00 Id 0 Lun: 0
                scsi5 Channel 00 Id 0 Lun: 1
                        Attached scsi disk sdb          State: running
```

3.2. Change directory to the location of the iSCSI node records for the remainder of this exercise. Locate the persistent node record for the new iSCSI target.

```
[root@desktop1 ~]# cd /var/lib/iscsi/nodes
[root@desktop1 nodes]$ ls -lR
```

3.3. View the connection parameters defaults in an iSCSI node record.

```
[root@desktop1 nodes]# less
 iqn.2014-06.com.example:server1/192.168.0.101,3260,1/default
```

4. Connect to and disconnect from the target, using both manual commands and methods that utilize the node record(s) to connect automatically.

4.1. Disconnect the iSCSI block device by logging out of the iSCSI target. Confirm that the corresponding iSCSI block device has disappeared.

Important

In this exercise, the block device was not formatted or otherwise used. If the device is in use, unmount it properly before continuing.

```
[root@desktop1 nodes]# iscsiadm -m node -T iqn.2014-06.com.example:server1 -p
 192.168.0.101 -u
[root@desktop1 nodes]$ lsblk
```

4.2. Confirm that the node record for the disconnected iSCSI target still exists.

```
[root@desktop1 nodes]# ls -lR
```

4.3. Restart the **iscsi** client service. Confirm that the iSCSI block device returns. The **iscsi** performs logins for all node records found.

```
[root@desktop1 nodes]# systemctl restart iscsi
[root@desktop1 nodes]# lsblk
```

4.4. Disconnect the iSCSI block device again by logging out of the iSCSI target. Additionally, delete the node record using the proper command. Confirm that the iSCSI block device has again disappeared.

```
[root@desktop1 nodes]# iscsiadm -m node -T iqn.2014-06.com.example:server1 -p
 192.168.0.101 -u
[root@desktop1 nodes]# iscsiadm -m node -T iqn.2014-06.com.example:server1 -p
 192.168.0.101 -o delete
[root@desktop1 nodes]# lsblk
```

4.5. Confirm that the node record for the disconnected iSCSI target no longer exists.

```
[root@desktop1 nodes]# ls -lR
```

4.6. Restart the **iscsi** client service. Confirm that the iSCSI block device does not return since there is no node record to trigger the target login.

```
[root@desktop1 nodes]# systemctl restart iscsi
[root@desktop1 nodes]# lsblk
```

5. Rediscover the target and confirm that the discovery step alone creates the node record(s).

5.1. Rediscover the configured iSCSI targets provided by a specific iSCSI target server portal, but do *not* log into the target.

```
[root@desktop1 nodes]# iscsiadm -m discovery -t st -p 192.168.0.101
192.168.0.101:3260,1 iqn.2014-06.com.example:server1
```

5.2. Confirm that the node record for the iSCSI target is created by discovery, even before login has occurred.

```
[root@desktop1 nodes]# ls -R
```

5.3. Clean up by deleting the node record again. Confirm that the node record no longer exists. When finished, return to your home directory.

```
[root@desktop1 nodes]# iscsiadm -m node -T iqn.2014-06.com.example:server1 -p
 192.168.0.101 -o delete
[root@desktop1 nodes]# ls -lR
[root@desktop1 nodes]# cd ~
```

Summary

iSCSI Concepts

In this section, students learned how to:

- Describe the history of the SCSI protocol and adaptation for use over networks.
- Describe block storage cabling topologies.
- Define iSCSI components and terms.

Providing iSCSI Targets

In this section, students learned how to:

- Identify the steps to install and configure an iSCSI target server.
- Describe the components for building a single iSCSI target.
- Describe the components for providing access security for each iSCSI target.

Accessing iSCSI Storage

In this section, students learned how to:

- Describe the installation and configuration of an iSCSI initiator.
- Outline the steps to access and login to an iSCSI LUN.
- Describe how iSCSI persistent connections are implemented.

CHAPTER 7

PROVIDING FILE-BASED STORAGE

Overview	
Goal	To provide NFS exports and SMB file shares to specific systems and users.
Objectives	• Export file systems to client systems using NFS, controlling access by IP address. • Export file systems to clients using NFS, controlling access with Kerberos and using labeled NFS. • Share file systems with clients using SMB, controlling access by username and password. • Mount an SMB share with the **`multiuser`** mount option, using password-based authentication and **`cifscreds`** to control access
Sections	• Exporting NFS File Systems (and Practice) • Protecting NFS Exports (and Practice) • Providing SMB File Shares (and Practice) • Performing a Multiuser SMB Mount (and Practice)
Lab	• Providing File-based Storage

Exporting NFS File Systems

Objectives

After completing this section, students should be able to export file systems to client systems using NFS, controlling access by IP address.

What is NFS?

The Network File System (NFS) is a network file system commonly used by UNIX systems and network-attached storage devices to allow multiple clients to share access to files over the network. It provides access to shared directories or files from client systems.

Warning

The NFS protocol transmits data in clear text over the network. Furthermore, the server relies on the client to identify users. It is not recommended to export directories with sensitive information without the use of Kerberos authentication and encryption, which is covered later in this section.

NFS exports

A NFS server installation requires the *nfs-utils* package to be installed. It provides all necessary utilities to export a directory with NFS to clients. The configuration file for the NFS server exports is the **`/etc/exports`** file.

The **`/etc/exports`** file lists the directory to share to client hosts over the network and indicates which hosts or networks have access to the export.

Note

Instead of adding the information required for exporting directories to the **`/etc/exports`** file, a newly created file named **`*.exports`** can be added to the **`/etc/exports.d/`** directory holding the configuration of exports.

Warning

Exporting the same directory with NFS and Samba is not supported on Red Hat Enterprise Linux 7, because NFS and Samba use different file locking mechanisms, which can cause file corruption.

One or more clients can be listed, separated by a space, as a:

- DNS-resolvable host name, like **`server0.example.com`** in the following example, where the **`/myshare`** directory is exported and can be mounted by **`server0.example.com`**.

```
/myshare server0.example.com
```

- DNS-resolvable host name with the wildcards ***** for multiple characters and/or **?** for a single character. The following example allows all subdomains in the **example.com** domain to access the NFS export.

```
/myshare *.example.com
```

- DNS-resolvable host name with character class lists in square brackets. In this example, the hosts **server0.example.com**, **server1.example.com**, ... , and **server20.example.com** have access to the NFS export.

```
/myshare server[0-20].example.com
```

- IPv4 address. The following example allows access to the **/myshare** NFS share from the **192.168.0.1** IP address.

```
/myshare 192.168.0.1
```

- IPv4 network. This example shows an **/etc/exports** entry, which allows access to the NFS-exported directory **/myshare** from the **192.168.0.0/16** network.

```
/myshare 192.168.0.0/16
```

- IPv6 address without square brackets. The following example allows the client with IPv6 address **2000:472:18:b51:c32:a21** access to the NFS-exported directory **/myshare**.

```
/myshare 2000:472:18:b51:c32:a21
```

- IPv6 network without square brackets. This example allows the IPv6 network **2000:472:18:b51::/64** access to the NFS export.

```
/myshare 2000:472:18:b51::/64
```

- A directory can be exported to multiple hosts simultaneously by specifying multiple targets with their options, separated by spaces, following the directory to export.

```
/myshare *.example.com 192.168.0.0/16
```

Optionally, there can be one or more export options specified in round brackets as a comma-separated list, directly followed by each client definition. Commonly used export options are:

- **ro**, read-only: the default setting when nothing is specified. It is allowed to explicitly specify it with an entry. Restricts the NFS clients to read files on the NFS share. Any write operation is prohibited. The following example explicitly states the **ro** flag for the server0.example.com host.

```
/myshare desktop0.example.com(ro)
```

- **rw**, read-write: allows read and write access for the NFS clients. In the following example, the **desktop0.example.com** is able to access the NFS export read-only, while **server[0-20].example.com** has read-write access to the NFS share.

```
/myshare desktop0.example.com(ro) server[0-20].example.com(rw)
```

- **no_root_squash**: By default, **root** on an NFS client is treated as user **nfsnobody** by the NFS server. That is, if **root** attempts to access a file on a mounted export, the server will treat it as an access by user **nfsnobody** instead. This is a security measure that can be problematic in scenarios where the NFS export is used as **/** by a diskless client and **root** needs to be treated as **root**. To disable this protection, the server needs to add **no_root_squash** to the list of options set for the export in **/etc/exports**.

  The following example allows the client **diskless.example.com** read-write and real root user access to the exported NFS directory **/myshare**.

```
/myshare diskless.example.com(rw,no_root_squash)
```

Warning

Note that this particular configuration is not secure, and would be better done in conjunction with Kerberos authentication and integrity checking, which is covered later in this section.

Configuring an NFS export

In this example, please follow along with these steps while your instructor demonstrates how to share a directory IP-based with NFS. The directory **/myshare** is on server1 and will be mounted on the desktop1 system.

1. Start the NFS service on server1 with the **systemctl** command.

```
[root@server1 ~]# systemctl start nfs-server
```

2. Enable the NFS service to start at boot on server1.

```
[root@server1 ~]# systemctl enable nfs-server
```

3. Create the directory **/myshare** to share it with NFS on the server1 system.

```
[root@server1 ~]# mkdir /myshare
```

4. Export the **/myshare** directory on server1 to the desktop1 client as a read- and write-enabled share. To do that, add the following line to the **/etc/exports** file on server1:

```
/myshare desktop1(rw)
```

5. After the changed **/etc/exports** file has been saved, apply the changes by executing **exportfs -r**.

```
[root@server1 ~]# exportfs -r
```

6. The NFS port 2049/TCP for **nfsd** must be open on the server. To configure **firewalld** to enable access to the NFS exports immediately, run:

```
[root@server1 ~]# firewall-cmd --permanent --add-service=nfs
```

7. Reload the firewalld rules so the new rule gets applied.

```
[root@server1 ~]# firewall-cmd --reload
```

8. Use the newly created mount point **/mnt/nfsexport** on the desktop1 system to mount the NFS-exported directory.

```
[root@desktop1 ~]# mkdir /mnt/nfsexport
```

9. On the desktop1 system, the share can now be mounted on the newly created mount point **/mnt/nfsexport** with the mount command.

```
[root@desktop1 ~]# mount server1:/myshare /mnt/nfsexport
```

References

nfs(5), **mount**(8), **mount.nfs**(8), **exportfs**(8), and **exports**(5) man pages

Practice: Exporting NFS File Systems

Guided exercise

In this lab, you will use NFS to provide shared storage.

Resources:	
Machines:	desktop1 and server1

Outcomes:
The NFS server exports an IP-based NFS share on server1. The NFS export is mounted on desktop1.

Before you begin...

- Reset the server1 system.
- Become **root** on your **server1** system.

```
[student@server1 ~]$ su
```

- Become **root** on your **desktop1** system.

```
[student@desktop1 ~]$ sudo -i
```

Configure an IP-based NFS share on server1 according to the following requirements:

- The NFS server provides newly created shared directory **/nfsshare**.
- The **/nfsshare** NFS export provides read and write access for **nfsnobody**.

Mount the share on the **/mnt/nfsshare** mount point on desktop1 permanently.

1. Configure an IP-based NFS share on server1 that provides a newly created shared directory **/nfsshare** for the desktop1 machine with read and write access for **nfsnobody**.

   1.1. Start the NFS service on server1.

   ```
   [root@server1 ~]# systemctl start nfs-server
   ```

   1.2. Enable the NFS service to start at boot on server1.

   ```
   [root@server1 ~]# systemctl enable nfs-server
   ```

   1.3. Create the directory **/nfsshare** to be shared by NFS on the server1 system.

   ```
   [root@server1 ~]# mkdir /nfsshare
   ```

   1.4. Change the ownership on the **/nfsshare** to user **nfsnobody**, so the directory is writable by **nfsnobody**.

```
[root@server1 ~]# chown nfsnobody /nfsshare
```

1.5. Change the **/etc/exports** configuration file on server1 to share the newly created **/nfsshare** directory on the desktop1 system with read and write access.

```
[root@server1 ~]# echo '/nfsshare desktop1(rw)' >>/etc/exports
```

1.6. Use the **exportfs -r** command to reload the **/etc/exports** configuration file on server1.

```
[root@server1 ~]# exportfs -r
```

1.7. Configure firewalld to allow access to the NFS service on server1.

```
[root@server1 ~]# firewall-cmd --permanent --add-service=nfs
```

1.8. Reload the **firewalld** configuration to allow access to the NFS service instantly on server1.

```
[root@server1 ~]# firewall-cmd --reload
```

2. Mount the NFS export from the server1 system on the **/mnt/nfsshare** mount point on desktop1 permanently.

2.1. Create the mount point **/mnt/nfsshare** on the desktop1 system.

```
[root@desktop1 ~]# mkdir /mnt/nfsshare
```

2.2. Create the required entry in **/etc/fstab** to mount the exported NFS share on the newly created **/mnt/nfsshare** directory on the desktop1 system permanently.

```
server1:/nfsshare /mnt/nfsshare nfs defaults 0 0
```

2.3. Mount the exported NFS share on the newly created **/mnt/nfsshare** directory on the desktop1 system and verify the **/etc/fstab** entry works as expected.

```
[root@desktop1 ~]# mount -a
```

2.4. Verify that the NFS share mounted at **/mnt/nfsshare** is writable on the desktop1 system.

```
[root@desktop1 ~]# touch /mnt/nfsshare/test.txt
[root@desktop1 ~]# ls -l /mnt/nfsshare
total 0
-rw-r--r--. 1 nfsnobody nfsnobody 0 May  8 04:14 test.txt
```

3. Unmount **/mnt/nfsshare** and remove the entry from **/etc/fstab**.

Protecting NFS Exports

Objectives

After completing this section, students should be able to export file systems to clients using NFS, controlling access with Kerberos and using labeled NFS.

Kerberos-enabled NFS exports

The NFS server does not require authentication and only enforces access restriction based on the IP address or host name of the client by default. To remedy this, the NFS server provides options to secure access to files using a number of methods: **none**, **sys**, **krb5**, **krb5i**, and **krb5p**. The NFS server can choose to offer a single method or multiple methods for each exported share. NFS clients must connect to the exported share using one of the methods mandated for that share, specified as a mount option **sec=*method***.

Security Methods

- **none**: Anonymous access to the files, writes to the server will be allocated UID and GID of **nfsnobody**. This requires the SELinux Boolean **nfsd_anon_write** to be active.
- **sys**: File access based on standard Linux file permissions for UID and GID values. If not specified, this is the default. The NFS server trusts any UID sent by the client.
- **krb5**: Clients must prove identity using Kerberos and then standard Linux file permissions apply. UID/GID is determined based upon the Kerberos principal from the accessing user.
- **krb5i**: Adds a cryptographically strong guarantee that the data in each request has not been tampered with. UID/GID is determined based upon the Kerberos principal from the accessing user.
- **krb5p**: Adds encryption to all requests between the client and the server, preventing data exposure on the network. This will have a performance impact, but provides the most security. UID/GID is determined based upon the Kerberos principal from the accessing user.

For using any of the security options that use a Kerberos server, the **nfs-secure-server** needs to be running in addition to the **nfs-server** service on the system exporting the NFS shares. The client requires the **nfs-secure** service to run to help with negotiating Kerberos authentication.

Important

Kerberos options will require, at a minimum, a **/etc/krb5.keytab** and additional authentication configuration that is not covered in this section, joining the Kerberos realm. The **/etc/krb5.keytab** will normally be provided by the authentication or security administrator. Request a **keytab** that includes either a *host principal*, *nfs principal*, or both.

Configuring a Kerberos-enabled NFS server

In this example, please follow along with these steps while your instructor demonstrates how to export a directory with NFS using krb5p security. The required keytab files are provided. The NFS

server exports the **/securedexport** directory to all subdomains of example.com. The client mounts the NFS export on the **/mnt/securedexport** mount point.

1. Install the keytab provided at `http://classroom.example.com/pub/keytabs/server1.keytab` on the server1 system, which will act as the NFS server.

```
[root@server1 ~]# wget -O /etc/krb5.keytab http://classroom.example.com/pub/keytabs/
server1.keytab
```

2. For NFS with Kerberos security, the **nfs-secure-server** needs to be running. Start the **nfs-secure-server** service on the server1 system.

```
[root@server1 ~]# systemctl start nfs-secure-server
```

3. Enable the **nfs-secure-server** to start at system boot on server1.

```
[root@server1 ~]# systemctl enable nfs-secure-server
```

4. Create the directory **/securedexport** on server1. This directory will be used as the NFS export.

```
[root@server1 ~]# mkdir /securedexport
```

5. Add the directory **/securedexport** to the **/etc/exports** file to export it with NFS. Enable krb5p security to secure access to the NFS share. Allow read and write access to the exported directory from all subdomains of the **example.com** domain.

```
[root@server1 ~]# echo '/securedexport *.example.com(sec=krb5p,rw)' >>/etc/exports
```

6. After the changed **/etc/exports** file has been saved, apply the changes by executing **exportfs -r**.

```
[root@server1 ~]# exportfs -r
```

7. The NFS port 2049/TCP for **nfsd** must be open on the server. To configure **firewalld** to enable access to the NFS exports immediately, run:

```
[root@server1 ~]# firewall-cmd --permanent --add-service=nfs
```

8. Reload the firewalld rules so the new rule gets applied.

```
[root@server1 ~]# firewall-cmd --reload
```

9. Install the provided keytab on the desktop1 system, which will act as the NFS client. Mount the krb5p-secured share on the desktop1 system:

```
[root@desktop1 ~]# wget -O /etc/krb5.keytab http://classroom.example.com/pub/
keytabs/desktop1.keytab
```

10. NFS uses the **nfs-secure** service on the client side to help negotiate and manage communication with the server when connecting to Kerberos-secured shares. It must be running to use the secured NFS shares; **start** and **enable** it to ensure it is always available on the desktop1 system.

```
[root@desktop1 ~]# systemctl enable nfs-secure
ln -s '/usr/lib/systemd/system/nfs-secure.service'  ...
[root@desktop1 ~]# systemctl start nfs-secure
```

Note

The **nfs-secure** service is part of the **nfs-utils** package, which should be installed by default. If it is not installed, run:

```
[root@desktop1 ~]# yum -y install nfs-utils
```

11. The mount point must exist to mount the krb5p-secured export from server1 on the desktop1 system. **/mnt/securedexport** is created on the desktop1 system.

```
[root@desktop1 ~]# mkdir /mnt/securedexport
```

12. The exported directory now can be mounted on the desktop1 system with krb5p security enabled.

```
[root@desktop1 ~]# mount -o sec=krb5p server1:/securedexport /mnt/securedexport
```

13. To clean up for the next demonstration, the **/mnt/securedexport** mount point needs to be unmounted on the desktop1 system.

```
[root@desktop1 ~]# umount /mnt/securedexport
```

SELinux and labeled NFS

SELinux offers additional security by locking down the capabilities of services provided in Red Hat Enterprise Linux. By default, NFS mounts have the SELinux context **nfs_t**, independent of the SELinux context they have on the server that provides the export.

This behavior can be changed on the client side by using the mount option **context="selinux_context"**. The following example mounts the NFS export and enforces the SELinux context: **system_u:object_r:public_content_rw_t:s0**:

```
[root@server1 ~]# mount -o context="system_u:object_r:public_content_rw_t:s0" server1:/
myshare /mnt/nfsexport
```

The NFS server can be forced to properly export the SELinux context of a share by switching to NFS version 4.2. This specification currently only exists as an Internet draft. It is already implemented in the NFS server shipped by Red Hat Enterprise Linux 7, but needs to be turned on explicitly.

To enable NFS version 4.2 on the server1 system to export the SELinux labels, change the **RPCNFSDARGS=""** line in the **/etc/sysconfig/nfs** file to:

```
RPCNFSDARGS="-V 4.2"
```

The **nfs-server** or **nfs-secure-server** respectively require a restart.

```
[root@server1 ~]# systemctl restart nfs-server
[root@server1 ~]# systemctl restart nfs-secure-server
```

On the client side, **mount -o v4.2** must be specified as the mount option.

```
[root@desktop1 ~]# mount -o sec=krb5p,v4.2 server1:/securedexport /mnt/securedexport
```

For testing purposes, a new file with the name **selinux.txt** is created in the exported directory **/securedexport**. After creation, the SELinux type is changed to **public_content_t**.

```
[root@server1 ~]# touch /securedexport/selinux.txt
[root@server1 ~]# chcon -t public_content_t /securedexport/selinux.txt
```

All SELinux labels are now properly handled by server1 and forwarded to the client system desktop1.

```
[root@desktop1 ~]# ls -Z /mnt/securedexport/
-rw-r--r--. root root unconfined_u:object_r:public_content_t:s0 selinux.txt
```

Note

In a default installation of Red Hat Enterprise Linux 7, the **nfs_export_all_ro** and **nfs_export_all_rw** SELinux Booleans are both enabled. This allows the NFS daemon to read and write almost any file. To lock down the capabilities of the NFS server, disable these Booleans. For content to be readable by NFS, it should have the **public_content_t** or **nfs_t** SELinux context. For content to be both readable and writable, it should have the **public_content_rw_t** or **nfs_t** context. If the **public_content_rw_t** context is used, the **nfsd_anon_write** Boolean must be enabled to allow writes. Additional NFS-related SELinux information can be found in the **nfsd_selinux**(8) man page, which is provided by the *selinux-policy-devel* RPM package.

References

nfs(5), **mount**(8), **mount.nfs**(8), **exportfs**(8), **exports**(5), and **nfsd_selinux**(8) man pages

Practice: Protecting NFS Exports

Guided exercise

In this lab, you will use NFS to provide shared storage protected with Kerberos and SELinux.

Resources:	
Machines:	desktop1 and server1

Outcomes:
The NFS server exports a Kerberized NFS share on server1 with SELinux labels. The NFS export is mounted on desktop1 with krb5p security and SELinux labels available.

Before you begin...

- Reset the server1 system.
- Become **root** on your **server1** system.

```
[student@server1 ~]$ su
```

- Set up your server system.

```
[root@server1 ~]# wget -O - http://instructor.example.com/pub/server1-nfssec.sh | bash
```

- Become **root** on your **desktop1** system.

```
[student@desktop1 ~]$ sudo -i
```

  Set up your **desktop1** system.

```
[root@desktop1 ~]# wget -O - http://instructor.example.com/pub/desktop1.nfssec.sh |
 bash
```

Configure the NFS server on server1 to meet the following requirements:

- Share the newly created **/securenfs** directory on server1 with krb5p security.
- Allow read and write access on the share from the desktop1 system.
- SELinux labels are exported.
- The NFS share is mounted on the **/mnt/secureshare** desktop1 with krb5p security and exported SELinux labels.
- Preconfigured krb5 keytabs for the server1 and desktop1 systems are available at:

  `http://classroom.example.com/pub/keytabs/server1.keytab`.

  `http://classroom.example.com/pub/keytabs/desktop1.keytab`.

1. Configure NFS to share the newly created **/securenfs** directory on server1 with krb5p security. Allow read and write access from the desktop1 system. The SELinux labels on the shared directory are exported.

   1.1. Install the keytab provided at `http://classroom.example.com/pub/keytabs/server1.keytab` on the server1 system.

   ```
   [root@server1 ~]# wget -O /etc/krb5.keytab http://classroom.example.com/pub/
   keytabs/server1.keytab
   ```

   1.2. Enable NFS version 4.2 on the server1 system to export the SELinux labels. To do that, change the **RPCNFSDARGS=""** line in the **/etc/sysconfig/nfs** file to:

   ```
   RPCNFSDARGS="-V 4.2"
   ```

   1.3. Start the nfs-secure-server service on the server1 system.

   ```
   [root@server1 ~]# systemctl restart nfs-secure-server
   ```

   1.4. Enable nfs-secure-server to start at system boot on server1.

   ```
   [root@server1 ~]# systemctl enable nfs-secure-server
   ```

   1.5. Create the directory **/securenfs** on server1.

   ```
   [root@server1 ~]# mkdir /securenfs
   ```

   1.6. Add the directory **/securenfs** to the **/etc/exports** file to export it with NFS. Enable krb5p security to secure access to the NFS share. Allow read and write access to the exported directory from the desktop1 system.

   ```
   [root@server1 ~]# echo '/securenfs desktop1(sec=krb5p,rw)' >>/etc/exports
   ```

   1.7. Reload the **/etc/exports** file on the server1 system.

   ```
   [root@server1 ~]# exportfs -r
   ```

   1.8. Configure firewalld to allow access to the NFS service on server1.

   ```
   [root@server1 ~]# firewall-cmd --permanent --add-service=nfs
   ```

   1.9. Reload the **firewalld** configuration to allow access to the NFS service instantly on server1.

   ```
   [root@server1 ~]# firewall-cmd --reload
   ```

2. Mount the krb5p-secured NFS share permanently on the **/mnt/secureshare** mount point so that all exported SELinux labels are present on the desktop1 system.

   2.1. Install the keytab provided at `http://classroom.example.com/pub/keytabs/desktop1.keytab` on the desktop1 system.

   ```
   [root@desktop1 ~]# wget -O /etc/krb5.keytab http://classroom.example.com/pub/
   keytabs/desktop1.keytab
   ```

   2.2. Start the nfs-secure service on desktop1 to help with negotiating authentication with a Kerberized NFS share.

   ```
   [root@desktop1 ~]# systemctl start nfs-secure
   ```

   2.3. Enable the nfs-secure service to start at system boot on desktop1.

   ```
   [root@desktop1 ~]# systemctl enable nfs-secure
   ```

   2.4. Create the mount point **/mnt/secureshare** on the desktop1 system.

   ```
   [root@desktop1 ~]# mkdir /mnt/secureshare
   ```

   2.5. Create the entry in the **/etc/fstab** file to mount the **/securenfs** share exported by the server1 system on the **/mnt/secureshare** mount point on desktop1 so that the SELinux labels from the share are shown on the mount point.

   ```
   server1:/securenfs /mnt/secureshare nfs defaults,v4.2,sec=krb5p 0 0
   ```

   2.6. Mount the exported NFS share on the newly created **/mnt/secureshare** directory on the desktop1 system and verify the **/etc/fstab** entry works as expected.

   ```
   [root@desktop1 ~]# mount -a
   ```

3. Test the setup with the newly created file **/securenfs/testfile.txt** with the content "Hello World" on the server1 machine. Set the SELinux context to **public_content_t** on the file **/securenfs/testfile.txt** on server1. Change the ownership of the **/securenfs/testfile.txt** file to ldapuser1:ldapuser1 and the permissions to 644. Verify that the SELinux context and the permissions are present on the mounted share on the desktop1 system. Verify that user ldapuser1 has read and write access on the **/mnt/secureshare/testfile.txt** file on desktop1.

   3.1. Create a new file **/securenfs/testfile.txt** with the content "Hello World" on the server1 machine.

   ```
   [root@server1 ~]# echo "Hello World" > /securenfs/testfile.txt
   ```

   3.2. Set the SELinux context to **public_content_t** on the file **/securenfs/testfile.txt** on server1.

```
[root@server1 ~]# chcon -t public_content_t /securenfs/testfile.txt
```

3.3. Change the ownership of the **/securenfs/testfile.txt** file to ldapuser1:ldapuser1 on server1.

```
[root@server1 ~]# chown ldapuser1:ldapuser1 /securenfs/testfile.txt
```

3.4. Change the permissions of the **/securenfs/testfile.txt** file to 644 on the server1 system.

```
[root@server1 ~]# chmod 644 /securenfs/testfile.txt
```

3.5. Verify the SELinux context is exported on the desktop1 system and available on the mounted share.

```
[root@desktop1 ~]# ls -Z /mnt/secureshare
-rw-r--r--. ldapuser1 ldapuser1 unconfined_u:object_r:public_content_t:s0
 testfile.txt
```

3.6. Log into the desktop1 system as ldapuser1 with password **kerberos** by using ssh.

```
[root@desktop1 ~]# ssh ldapuser1@desktop1
...
ldapuser1@desktop1's password: kerberos
Creating home directory for ldapuser1
```

3.7. Verify the file **/mnt/secureshare/testfile.txt** is writable by the Kerberos-authenticated ldapuser1.

```
[ldapuser1@desktop1 ~]$ echo "I can write" >>/mnt/secureshare/testfile.txt
[ldapuser1@desktop1 ~]$ cat /mnt/secureshare/testfile.txt
Hello World
I can write
```

3.8. Unmount the NFS filesystem and remove the entry from /etc/fstab

```
[root@desktop1 ~]# umount /mnt/secureshare
```

```
[root@desktop1 ~]# vi /etc/fstab
```

Providing SMB File Shares

Objectives

After completing this section, students should be able to share directories with clients using SMB, controlling access by username and password.

What is SMB?

Server Message Block (SMB) is the standard file-sharing protocol for Microsoft Windows servers and clients. SMB file servers can be configured in a number of different ways. One of the simplest is to configure the file servers and their clients as members of a common Windows *workgroup*, which announces servers and clients to the local subnet. The file servers each manage their own local user accounts and passwords independently. More sophisticated configurations may be members of a *Windows domain* and coordinate user authentication through a *domain controller*.

Using a software package named Samba, Red Hat Enterprise Linux is able to act as a server for SMB file shares. Mounting SMB file shares as a client is handled by a kernel driver and utilities included in the *cifs-utils* package.

SMB file sharing with Samba

The Samba service can share Linux file systems as SMB network file shares. This section will cover the basic configuration steps needed for a Samba server to provide a file share to the members of a Windows workgroup, managing its own users locally. It will not discuss the more complex configuration required to make the Samba server a member of a Windows domain.

The basic steps that must be performed in order to configure Samba to provide an SMB file share as a workgroup member are:

1. Install the *samba* package.
2. Prepare the permissions on the directory to be shared.
3. Configure **`/etc/samba/smb.conf`**.
4. Set up appropriate Linux users with NTLMv2 passwords.
5. Start Samba and open the local firewall.
6. Verify that the share can be mounted from a client.

Installing Samba

To deploy the Samba service on a Red Hat Enterprise Linux system, the *samba* package must be installed. This can be done directly, or as part of the *file-server* package group:

```
[root@server1 ~]# yum install samba
```

Preparing directories for sharing

Warning

Do not use Samba to share a directory that is also an NFS export or a mounted NFS file system. This can result in file corruption, stale file locks, or other file access issues with the share.

The directory to be shared must be created if it does not already exist.

```
[root@server1 ~]# mkdir /sharedpath
```

Users and regular permissions
The permissions which should be set on the directory will depend on who needs access to it and how it will be mounted by clients.

A client normally mounts a share by authenticating access to the SMB server as a particular user. All files on the share need to be readable (and possibly writable) by the user that is used to mount the share.

SELinux contexts and Booleans
In order for Samba to work correctly when SELinux is in enforcing mode, the directory will need to have correct SELinux contexts and certain SELinux Booleans may need to be set.

If the shared directory will only be accessed through Samba, then the directory and all its subdirectories and files should be labeled **samba_share_t**, which gives Samba read and write access. It is best to configure the SELinux policy so that **restorecon** will set this type on the share and its contents if the file system is relabeled.

For example, to configure **restorecon** so that the files in **/sharedpath** have the type **samba_share_t**, and then to relabel the directory, run:

```
[root@server1 ~]# semanage fcontext -a -t samba_share_t '/sharedpath(/.*)?'
[root@server1 ~]# restorecon -vvFR /sharedpath
restorecon reset /sharedpath context unconfined_u:object_r:default_t:s0-
>system_u:object_r:samba_share_t:s0
```

Note

Samba can also serve files labeled with the SELinux types **public_content_t** (read-only) and **public_content_rw_t** (read-write). To allow read-write access to files and directories labeled **public_content_rw_t**, the SELinux Boolean **smbd_anon_write** must also be enabled.

Configuring /etc/samba/smb.conf

The main configuration file for Samba is **/etc/samba/smb.conf**. This file is divided into multiple sections. Each section starts with a section name in square brackets, followed by a list of parameters set to particular values.

/etc/samba/smb.conf starts with a **[global]** section used for general server configuration. Subsequent sections each define a file share or printer share provided by the Samba server. Two special sections may exist, **[homes]** and **[printers]**, which have special uses. Any line beginning with either a semicolon (**;**) or a hash (**#**) character is commented out.

The [global] section

The **[global]** section defines the basic configuration of the Samba server. There are three things which should be configured here:

1. **workgroup** is used to specify the Windows workgroup for the server. Most Windows systems default to **WORKGROUP**, although Windows XP Home defaulted to **MSHOME**. This is used to help systems browse for the server using the NetBIOS for TCP/IP name service.

   To set the workgroup to **WORKGROUP**, change the existing workgroup entry in the **/etc/samba/smb.conf** to:

   ```
   workgroup = WORKGROUP
   ```

2. **security** controls how clients are authenticated by Samba. For **security = user**, clients log in with a valid username and password managed by the local Samba server. This setting is the default in **/etc/samba/smb.conf**.

3. **hosts allow** is a comma-, space-, or tab-delimited list of hosts that are permitted to access the Samba service. If it is not specified, all hosts can access Samba. If it is not specified in the **[global]** section, it can be set on each share separately. If it is specified in the **[global]** section, then it will apply to all shares, regardless of whether each share has a different setting.

   Hosts can be specified by host name or by source IP address. Host names are checked by reverse-resolving the IP address of the incoming connection attempt. The full syntax of this directive is described by the **hosts_access**(5) man page.

   Allowed hosts can be specified in a number of ways:

   - IPv4 network/prefix: **192.168.0.0/24**
   - IPv4 network/netmask: **192.168.0.0/255.255.255.0**
   - If the IPv4 subnet prefix is on a byte boundary: **192.168.0.**
   - IPv6 network/prefix: **[2001:db8:0:1::/64]**
   - Host name: **desktop.example.com**
   - All hosts ending in example.com: **.example.com**

   For example, to restrict access to only the hosts from the **192.168.0.0/16** network using the *trailing dot* notation, the **hosts allow** entry in the **/etc/samba/smb.conf** configuration file would read:

   ```
   hosts allow = 192.168.
   ```

   To additionally allow access from all host names ending with ".example.com", the **/etc/samba/smb.conf** configuration file entry would be:

```
hosts allow = 192.168. .example.com
```

File share sections

To create a file share, at the end of **/etc/samba/smb.conf**, place the share name in brackets to start a new section for the share. Some key directives should be set in this section:

1. **path** must be set to indicate which directory to share; for example, **path = /sharedpath**.

2. **writable = yes** should be set if all authenticated users should have read-write access to the share. The default setting is **writable = no**.

   If **writable = no** is set, a comma-separated **write list** of users with read-write access to the share can be provided. Users not on the list will have read-only access. Members of local groups can also be specified: **write list = @management** will permit all authenticated users who are members of the Linux group "management" to have write access.

3. **valid users**, if set, specifies a list of users allowed to access the share. Users not on the list are not allowed to access the share. However, if the list is blank, *all* users can access the share.

For example, to allow only user **fred** and members of group **management** read-only access to the share **myshare**, the section would read:

```
[myshare]
    path = /sharedpath
    writable = no
    valid users = fred, @management
```

The [homes] section

The **[homes]** section defines a special file share, which is enabled by default. This share makes local home directories available via SMB. The share name can be specified as **homes**, in which case the Samba server will convert it to the home directory path of the authenticating user, or as a specific ***username***.

Important

The **samba_enable_home_dirs** SELinux Boolean allows local Linux home directories to be shared by Samba to other systems. This needs to be enabled for **[homes]** to work (**setsebool -P samba_enable_home_dirs=on**).

The **use_samba_home_dirs** Boolean, on the other hand, allows remote SMB file shares to be mounted and used as local Linux home directories. It is easy to confuse the two options.

Validating /etc/samba/smb.conf

To verify that there are no errors in the edited **smb.conf** file, the command **testparm** is available. Run **testparm** with no arguments to verify that there are no obvious syntax errors.

```
[root@server1 ~]# testparm
```

```
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section "[random]"
Processing section "[homes]"
Processing section "[printers]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions

[global]
 server string = Samba Server Version %v
 log file = /var/log/samba/log.%m
 max log size = 50
 idmap config * : backend = tdb
 cups options = raw

[random]
 comment = Test File Share
 path = /srv/random

[homes]
 comment = Home Directories
 read only = No
 browseable = No

[printers]
 comment = All Printers
 path = /var/spool/samba
 printable = Yes
 print ok = Yes
 browseable = No
```

Important

The directive **`read only = no`** is the same as **`writable = yes`**, which can be confusing.

Preparing Samba users

The **`security = user`** setting requires a Linux account with a Samba account that has a valid NTLM password. To create a Samba-only system user, keep the Linux password locked, and set the login shell to **`/sbin/nologin`**. This prevents the login of the user directly, or with **`ssh`** on the system.

For example, to create the locked Linux account for a user **`fred`**:

```
[root@server1 ~]# useradd -s /sbin/nologin fred
```

The *samba-client* contains the **`smbpasswd`** command. It can create Samba accounts and set passwords.

```
[root@server1 ~]# yum -y install samba-client
```

If **`smbpasswd`** is passed a username without any options, it will attempt to *change* the account password. The root user can use it with the **`-a`** option to add the Samba account and set the NTLM password. The **`-x`** option can be used by root to delete a Samba account and password for a user.

For example, to create a Samba account for user **fred** and assign an NTLM password:

```
[root@server1 ~]# smbpasswd -a fred
New SMB password:  redhat
Retype new SMB password:  redhat
...
Added user fred.
```

A more powerful tool than **smbpasswd** is also available for the **root** user, **pdbedit**. For example, **pdbedit -L** will list all users with Samba accounts configured on the system. For more information, see the **pdbedit**(8) man page.

Starting Samba

Use **systemctl** to start the Samba services immediately and enable them to start at boot time:

```
[root@server1 ~]# systemctl start smb nmb
[root@server1 ~]# systemctl enable smb nmb
```

The two services these units start, **smbd** and **nmbd**, must communicate through the local firewall. Samba's **smbd** daemon normally uses TCP/445 for SMB connections. It also listens on TCP/139 for NetBIOS over TCP backward compatibility. The **nmbd** daemon uses UDP/137 and UDP/138 to provide NetBIOS over TCP/IP network browsing support.

To configure **firewalld** to allow clients to talk to the local Samba services, run:

```
[root@server1 ~]# firewall-cmd --permanent --add-service=samba
success
[root@server1 ~]# firewall-cmd --reload
success
```

Important

Samba checks periodically to determine if **/etc/samba/smb.conf** has been changed. If the configuration file has changed, Samba automatically reloads it. This will not affect any connections already established to the Samba service, until the connection is closed or Samba is completely restarted.

The command **systemctl reload smb nmb** can be used to reload the configuration file immediately, or **systemctl restart smb nmb** to restart Samba entirely.

Mounting SMB file systems

Regular SMB mounts

The *cifs-utils* package can be used to mount SMB file shares on the local system, whether from a Samba server or a native Microsoft Windows server. By default, SMB mounts use a single set of user credentials (the *mount credentials*) for mounting the share and determining access rights to files on the share. All users on the Linux system using the mount use the same credentials to determine file access.

The **mount** command is used to mount the share. By default, the protocol used to authenticate users is NTLMv2 password hashing encapsulated in Raw NTLMSSP messages (**sec=ntlmssp**),

as expected by recent versions of Microsoft Windows. The mount credentials can be provided in two ways. If mounting interactively at a shell prompt, the **username=** option can be used to specify which SMB user to authenticate as; the user will be prompted for the password. If mounting automatically, a credentials file readable only by root containing the username and password can be provided with the **credentials=** option.

For example, to mount the share **myshare** from the SMB file server **server1**, authenticating as SMB user **fred**, who has the NTLM password **redhat**:

```
[root@desktop1 ~]# mkdir /mnt/myshare
[root@desktop1 ~]# mount -o username=fred //server1/myshare /mnt/myshare
Password for fred@//server1/myshare:  redhat
```

References

samba(7), **smb.conf**(5), **testparm**(1), **mount**(8), **mount.cifs**(8), **smbpasswd**(8), **pdbedit**(8), and **samba_selinux**(8) man pages

Practice: Providing SMB File Shares

Guided exercise

In this lab, you will use SMB to provide shared storage.

Resources:	
Machines:	desktop1 and server1

Outcomes:
Share a directory with SMB on server1 according to the given requirements, then mount it on desktop1.

Before you begin...

- Reset the server1 system.
- Become **root** on your **server1** system.

```
[student@server1 ~]$ sudo -i
```

- Become **root** on your **desktop1** system.

```
[student@desktop1 ~]$ su
```

Configure a SMB share on the server1 system according to the following requirements:

- Share the newly created directory **/smbshare** with SMB.
- Members of the auxiliary group **marketing** have read and write permissions on the share.
- All users that are not member of the **marketing** group have read permission.
- The Samba server is in the **mycompany** workgroup and the share name in Samba is **smbshare**.
- Create the Samba-only user **brian**, who is part of the marketing team, with the password **redhat**.
- Create the new Samba-only user **rob** with the password **redhat**, who is not part of the marketing team.

1. Deploy the required RPM packages to run the SMB service on server1.

```
[root@server1 ~]# yum -y install samba
```

2. Create the auxiliary system group **marketing** and the **/smbshare** directory on server1. The **marketing** system group owns the **/smbshare** directory. Adjust the permissions on the **/smbshare** directory to have the SGID bit set, and write is prohibited by others. The SELinux context type on the **/smbshare** directory and all newly created files and subdirectories is **samba_share_t**.

2.1. Create the auxiliary system group **marketing** on the server1 system.

```
[root@server1 ~]# groupadd -r marketing
```

2.2. Create the **/smbshare** directory on server1.

```
[root@server1 ~]# mkdir -p /smbshare
```

2.3. Change the group ownership of the **/smbshare** directory to **marketing** on the server1 system.

```
[root@server1 ~]# chgrp marketing /smbshare
```

2.4. Adjust the permissions on the **/smbshare** directory to have the SGID bit set, and write is prohibited by others.

```
[root@server1 ~]# chmod 2775 /smbshare
```

2.5. Add the directory **/smbshare**, and all files shown as follows, to the SELinux policy as a directory sharing files with SMB by setting its label to **samba_share_t**.

```
[root@server1 ~]# semanage fcontext -a -t samba_share_t '/smbshare(/.*)?'
```

2.6. Apply the SELinux rule for the **/smbshare** directory that was added in the previous step on server1.

```
[root@server1 ~]# restorecon -vvFR /smbshare
restorecon reset /smbshare context unconfined_u:object_r:default_t:s0-
>system_u:object_r:samba_share_t:s0
```

3. Change the **/etc/samba/smb.conf** configuration file on server1 to reflect the configuration requested.

   Modify or confirm the following:

```
[global]
...
 workgroup = mycompany
...
 security = user
 passdb backend = tdbsam
```

   Add a section at the end of the file as follows.

```
[smbshare]
 path = /smbshare
 write list = @marketing
```

4. Create the Samba-only user **brian**, who is part of the marketing team. The user **brian** has read and write access to the **smbshare** SMB share. A new Samba user **rob** is created, who is not part of the marketing team. The user **rob** has read access to the **smbshare** SMB share. Both newly added users have the SMB password **redhat**.

   4.1. Install the *samba-client* RPM package because it contains **smbpasswd**.

   ```
   [root@server1 ~]# yum -y install samba-client
   ```

   4.2. Create the system user **brian** as a member of the auxiliary group **marketing** on server1.

   ```
   [root@server1 ~]# useradd -s /sbin/nologin -G marketing brian
   ```

   4.3. Add the SMB user **brian** to Samba. The Samba user is automatically mapped to the local system user **brian**.

   ```
   [root@server1 ~]# smbpasswd -a brian
   New SMB password:  redhat
   Retype new SMB password:  redhat
   Added user brian.
   ```

   4.4. Create the system user **rob** on server1.

   ```
   [root@server1 ~]# useradd -s /sbin/nologin rob
   ```

   4.5. Add the SMB user **rob** to Samba. The Samba user is automatically mapped to the local system user **rob**.

   ```
   [root@server1 ~]# smbpasswd -a rob
   New SMB password:  redhat
   Retype new SMB password:  redhat
   Added user rob.
   ```

5. Start and enable the **smb** and **nmb** services, and allow access to them through the firewall on server1.

   5.1. Start the **smb** and **nmb** services on the server1 system.

   ```
   [root@server1 ~]# systemctl start smb nmb
   ```

   5.2. Enable the **smb** and **nmb** services to start at system boot on server1.

   ```
   [root@server1 ~]# systemctl enable smb nmb
   ```

   5.3. Configure firewalld to allow access to the SMB service on server1.

   ```
   [root@server1 ~]# firewall-cmd --permanent --add-service=samba
   success
   ```

```
[root@server1 ~]# firewall-cmd --reload
success
```

6. Verify the newly created SMB share works as expected on the desktop1 system with the created Samba-only users **brian** and **rob**. The user **brian** has read and write access to the **smbshare** SMB share. The user **rob** has read access to the **smbshare** SMB share.

   6.1. Install the *cifs-utils* package because it provides the **mount.cifs** command.

```
[root@desktop1 ~]# yum -y install cifs-utils
```

   6.2. Create the mount point **/mnt/brian** on desktop1.

```
[root@desktop1 ~]# mkdir /mnt/brian
```

   6.3. Mount the **//server1/smbshare** Samba share temporarily as user **brian** on the mount point **/mnt/brian** on the desktop1 system.

```
[root@desktop1 ~]# mount -o username=brian //server1/smbshare /mnt/brian
Password for brian@//server1/smbshare:  redhat
```

   6.4. Verify on desktop1 that user **brian** has read and write access to the **smbshare** share provided by server1 because he is a member of the marketing auxiliary group.

```
[root@desktop1 ~]# echo "Hello World" >/mnt/brian/brian1.txt
[root@desktop1 ~]# cat /mnt/brian/brian1.txt
Hello World
```

   6.5. Create the mount point **/mnt/rob** on desktop1.

```
[root@desktop1 ~]# mkdir /mnt/rob
```

   6.6. Mount the **//server1/smbshare** Samba share temporarily as user **rob** on the mount point **/mnt/rob** on the desktop1 system.

```
[root@desktop1 ~]# mount -o username=rob //server1/smbshare /mnt/rob
Password for rob@//server1/smbshare:  redhat
```

   6.7. Verify on desktop1 that user **rob** has no write permission to the **smbshare** Samba share provided by the server1 system.

```
[root@desktop1 ~]# touch /mnt/rob/rob1.txt
touch: cannot touch `/mnt/rob/rob1.txt´: Permission denied
```

   6.8. Test if read access for Samba user **rob** works as expected on the **/mnt/rob** Samba share on the desktop1 system.

```
[root@desktop1 ~]# cat /mnt/rob/brian1.txt
```

```
Hello World
```

6.9. Unmount the SMB filesystems

```
[root@desktop1 ~]# umount /mnt/rob /mnt/brian
```

Note

Do not reset your server1 system. It will be used in the next lab.

Performing a Multiuser SMB Mount

Objectives

After completing this section, students should be able to mount an SMB share with the **multiuser** mount option, using password-based authentication and **cifscreds** to control access.

- Perform a multiuser SMB mount.
- Access the multiuser SMB mount.

Multiuser mounts with Samba

When a Samba share is mounted, the mount credentials determine the access permissions on the mount point by default. The new **multiuser** mount option separates the mount credentials from the credentials used to determine file access for each user. In Red Hat Enterprise Linux 7, this can be used with **sec=ntlmssp** authentication (contrary to the **mount.cifs**(8) man page).

The root user mounts the share using the **multiuser** option and an SMB username that has minimal access to the contents of the share. Regular users can then stash their own SMB usernames and passwords in the current session's kernel keyring with the **cifscreds** command. Their accesses to the share are authenticated with their own credentials from the keyring, not the mount credentials. The users can clear or change their credentials for that login session at any time, and they are cleared when the session ends. File access permissions are enforced entirely by the SMB server based on the access credentials currently in use.

For example, to create a new mount point **/mnt/multiuser** and mount the share **myshare** from the SMB file server **server1**, authenticating as SMB user **fred**, who has the NTLM password **redhat**, and using the **multiuser** mount option:

```
[root@desktop1 ~]# mkdir /mnt/multiuser
[root@desktop1 ~]# mount -o multiuser,sec=ntlmssp,username=fred \
> //server1/myshare /mnt/multiuser
Password for fred@//server1/myshare:  redhat
```

The command **cifscreds** is required to store authentication credentials in the keyring of the local user. Those authentication credentials are forwarded to the Samba server on a multiuser mount. The *cifs-utils* package provides the **cifscreds** command, so it is required on the desktop1 system.

```
[root@desktop1 ~]# yum -y install cifs-utils
```

The **cifscreds** command has various actions:

- **add** to add SMB credentials to the session keyring of a user. This option is followed by the host name of the SMB file server.
- **update** to update existing credentials in the session keyring of the user. This option is followed by the host name of the SMB file server.
- **clear** to remove a particular entry from the session keyring of the user. This option is followed by the host name of the Samba server.

- **clearall** to clear all existing credentials from the session keyring of the user.

Note

By default, **cifscreds** assumes that the username to use with the SMB credentials matches the current Linux username. A different username can be used for SMB credentials with the **-u *username*** option after the **add**, **update**, or **clear** action.

For example, assume that root has mounted **//server1/myshare** on the mount point **/mnt/multiuser** using the **multiuser** option. In order to access files on that share, user **frank** must use **cifscreds** to temporarily stash his username and password in the kernel-managed session keyring.

```
[frank@desktop1 ~]$ cifscreds add server1
Password:  redhat
[frank@desktop1 ~]$ echo "Frank was here" >/mnt/multiuser/frank2.txt
[frank@desktop1 ~]$ cat /mnt/multiuser/frank2.txt
Frank was here.
[frank@desktop1 ~]$ exit
```

Assume that the permissions on the files in the SMB share grant **frank** read-write access to the directory, but **jane** is only granted read access.

```
[jane@desktop1 ~]$ cifscreds add server1
Password:  redhat
[jane@desktop1 ~]$ echo "Jane was not here" >/mnt/multiuser/jane2.txt
-bash: /mnt/multiuser/jane2.txt: Permission denied
[jane@desktop1 ~]$ cat /mnt/multiuser/frank2.txt
Frank was here
```

References

mount(8), **mount.cifs**(8), and **cifscreds**(1) man pages

Practice: Performing a Multiuser SMB Mount

Guided exercise

In this lab, you will mount a SMB share with the multiuser option.

Resources:	
Machines:	desktop1 and server1

Outcomes:
The SMB share provided by server1 is mounted on the desktop1 system with the multiuser mount option.

Before you begin...
- Do not reset the server1 system.
- Become **root** on your **server1** system.

```
[student@server1 ~]$ sudo -i
```

- Become **root** on your **desktop1** system.

```
[student@desktop1 ~]$ sudo -i
```

Mount the SMB share **//server1/smbshare** permanently on the desktop1 system according to the following requirements:

- The mount point on the desktop1 system is the newly created directory **/mnt/multiuser**.
- The SMB share is mounted with a newly created credentials file **/root/smb-multiuser.txt**. The credentials used to mount the SMB share are username **brian** and password **redhat**.
- The SMB share is mounted with the multiuser mount option enabled.
- The already existing user **brian** on the desktop1 system has a corresponding SMB account on server1. Associate the system user **brian** on the desktop1 system with the SMB user **brian** on the server1 system to access the **/mnt/multiuser** mount point. The password for **brian** is **redhat**. Verify that user **brian** has read and write access to the mounted SMB share.
- The already existing user **rob** on the desktop1 system has a corresponding SMB account on server1. Associate the system user **rob** on the desktop1 system with the SMB user **rob** on the server1 system to access the **/mnt/multiuser** mount point. The password for **rob** is **redhat**. Verify that user **rob** has read but no write access to the mounted SMB share.

1. Install the *cifs-utils* RPM package on the desktop1 system because it contains the **cifscreds** command required to store and forward authentication credentials to the Samba server with a multiuser mount.

```
[root@desktop1 ~]# yum -y install cifs-utils
```

2. Mount the Samba share permanently on the **/mnt/multiuser** mount point on desktop1 and authenticate with a credentials file. Mount the Samba share with the credentials of user **brian**.

   2.1. Create the mount point **/mnt/multiuser** on desktop1.

   ```
   [root@desktop1 ~]# mkdir /mnt/multiuser
   ```

   2.2. Create the credentials file **/root/smb-multiuser.txt** with the username and password of user **brian** on the desktop1 system.

   ```
   [root@desktop1 ~]# echo 'username=brian' >/root/smb-multiuser.txt
   [root@desktop1 ~]# echo 'password=redhat' >>/root/smb-multiuser.txt
   ```

   2.3. Create the entry in **/etc/fstab** to permanently mount the Samba share with multiuser option as user brian on the **/mnt/multiuser** mount point on desktop1.

   ```
   //server1/smbshare /mnt/multiuser cifs credentials=/root/smb-
   multiuser.txt,multiuser,sec=ntlmssp 0 0
   ```

   2.4. Verify the entry in **/etc/fstab** to permanently mount the Samba share on desktop1 is correct by mounting the share with the fstab entry.

   ```
   [root@desktop1 ~]# mount /mnt/multiuser
   ```

3. Access the SMB multiuser mount **/mnt/multiuser** as the already existing user **brian** on desktop1. Automatically authenticate to Samba as the corresponding Samba user that exists with the same name on server1 and has read and write permission on the SMB share.

   3.1. Switch to user **brian** on the terminal on desktop1.

   ```
   [root@desktop1 ~]# useradd brian
   [root@desktop1 ~]# su - brian
   ```

   3.2. Try to write to the **/mnt/multiuser** mount point with user brian on the desktop1 system.

   ```
   [brian@desktop1 ~]$ touch /mnt/multiuser/testfile.txt
   touch: cannot touch `test.txt´: Permission denied
   ```

   3.3. Record the Samba credentials for the local user **brian** with the **cifscreds** command on desktop1.

   ```
   [brian@desktop1 ~]$ cifscreds add server1
   Password:  redhat
   ```

   3.4. Verify user **brian** has read and write permission on the mount point **/mnt/multiuser** on desktop1.

```
[brian@desktop1 ~]$ echo "Multiuser" >/mnt/multiuser/brian2.txt
[brian@desktop1 ~]$ cat /mnt/multiuser/brian2.txt
Multiuser
```

3.5. Exit the shell on the desktop1 system.

```
[brian@desktop1 ~]$ exit
[root@desktop1 ~]#
```

4. Access the SMB multiuser mount **/mnt/multiuser** as the already existing user **rob** on desktop1. Automatically authenticate to Samba as the corresponding Samba user **rob** that exists on server1 with the same name and has read permission on the SMB share.

4.1. Switch to user **rob** on the terminal.

```
[root@desktop1 ~]# useradd rob
[root@desktop1 ~]# su - rob
```

4.2. Record the Samba credentials for the local user **rob** with **cifscreds** on desktop1.

```
[rob@desktop1 ~]$ cifscreds add server1
Password:  redhat
```

4.3. Verify user **rob** has read but no write permission on the mount point **/mnt/multiuser** on desktop1.

```
[rob@desktop1 ~]$ echo "Multiuser" >/mnt/multiuser/rob2.txt
-bash: /mnt/multiuser/rob2.txt: Permission denied
[rob@desktop1 ~]$ cat /mnt/multiuser/brian2.txt
Multiuser
[rob@desktop1 ~]$ exit
```

5. Unmount the SMB filesystem.

```
[root@desktop1 ~]# umount /mnt/multiuser
```

Summary

Exporting NFS File Systems

In this section, students learned how to:

- Describe NFS server configuration.
- Export an IP-based NFS share.

Protecting NFS Exports

In this section, students learned how to:

- Set up Kerberos-enabled NFS.
- Configure labeled NFS and SELinux.

Providing SMB File Shares

In this section, students learned how to perform SMB server configuration.

Performing a Multiuser SMB Mount

In this section, students learned how to perform a multiuser SMB mount.

CHAPTER 8

CONFIGURING MARIADB DATABASES

Overview	
Goal	To provide a MariaDB SQL database for use by programs and database administrators.
Objectives	• Install MariaDB. • Configure and administer MariaDB. • Configure user and access rights. • Back up and restore MariaDB databases.
Sections	• Installing MariaDB (and Practice) • Working with MariaDB Databases (and Practice) • Managing Database Users and Access Rights (and Practice) • Creating and Restoring MariaDB Backups (and Practice)
Lab	• Configuring MariaDB Databases

Installing MariaDB

Objectives

After completing this section, students should be able to describe a relational database and be able to install and perform a basic configuration of a simple MariaDB relational database server.

Relational databases

A relational database is a mechanism that allows the persistence of data in an organized way. Databases store data items organized as a set of tables, with each table representing an entity. In a given table, each row corresponds to a record, while each column corresponds to an attribute of that record.

```
MariaDB [inventory]> SELECT * FROM product;
+----+-------------------+---------+-------+-------------+-----------------+
| id | name              | price   | stock | id_category | id_manufacturer |
+----+-------------------+---------+-------+-------------+-----------------+
|  1 | ThinkServer TS140 | 539.88  |    20 |           2 |               4 |
|  2 | ThinkServer TS440 | 1736.00 |    10 |           2 |               4 |
|  3 | RT-AC68U          | 219.99  |    10 |           1 |               3 |
|  4 | X110 64GB         |  73.84  |   100 |           3 |               1 |
+----+-------------------+---------+-------+-------------+-----------------+
4 rows in set (0.00 sec)
```

```
MariaDB [inventory]> SELECT * FROM category;
+----+------------+
| id | name       |
+----+------------+
|  1 | Networking |
|  2 | Servers    |
|  3 | Ssd        |
+----+------------+
3 rows in set (0.00 sec)
```

```
MariaDB [inventory]> SELECT * FROM manufacturer;
+----+----------+----------------+------------------+
| id | name     | seller         | phone_number     |
+----+----------+----------------+------------------+
|  1 | SanDisk  | John Miller    | +1 (941) 329-8855 |
|  2 | Kingston | Mike Taylor    | +1 (341) 375-9999 |
|  3 | Asus     | Wilson Jackson | +1 (432) 367-8899 |
|  4 | Lenovo   | Allen Scott    | +1 (876) 213-4439 |
+----+----------+----------------+------------------+
4 rows in set (0.00 sec)
```

The previous tables show:

- The **`product`** table has four records. Each record has six attributes: **`(id, name, price, stock, id_category, id_manufacturer)`**.
- **`X110 64GB`** is an SSD manufactured by SanDisk.
- The **`seller`** responsible for the ThinkServer TS140 product is Allen Scott.

There are two relational database packages provided with Red Hat Enterprise Linux 7:

- PostgreSQL—An open source database developed by the PostgreSQL Global Development Group, consisting of Postgres users (both individuals and companies) and other companies and volunteers, supervised by companies such as Red Hat and EnterpriseDB.
- MariaDB—A community-developed branch of MySQL built by some of the original authors of MySQL. It offers a rich set of feature enhancements, including alternate storage engines, server optimizations, and patches. The MariaDB Foundation works closely and cooperatively with the larger community of users and developers in the spirit of free and open source software.

Note

MariaDB is a community-developed replacement for MySQL. MySQL is not shipped in the core toolset for Red Hat Enterprise Linux 7, but it is available through Red Hat Software Collections 1.1.

Red Hat Software Collections is a fully-supported parallel set of tools included as part of most RHEL 7 subscriptions. Red Hat Software Collections provides a set of dynamic programming languages, database servers, web servers and various related packages that are either more recent than the version shipped in the core RHEL distribution or are not otherwise available in it at all. Packages in Red Hat Software Collections have a faster but shorter support cycle (up to 3 years of support).

More information is available in the *Red Hat Software Collections 1.1 Release Notes* at `https://access.redhat.com/docs`.

MariaDB installation

A full MariaDB database installation requires both the *mariadb* and *mariadb-client* groups of software to be installed.

The following packages will be installed with the **`mariadb`** group:

- *mariadb-server*—The MariaDB server and related files (mandatory package).
- *mariadb-bench*—MariaDB benchmark scripts and data (optional package).
- *mariadb-test*—The test suite distributed with MariaDB (optional package).

The following packages will be installed with the **`mariadb-client`** group:

- *mariadb*—A community-developed branch of MySQL (mandatory package).
- *MySQL-python*—A MariaDB interface for Python (default package).
- *mysql-connector-odbc*—ODBC driver for MariaDB (default package).
- *libdbi-dbd-mysql*—MariaDB plug-in for libdbi (optional package).
- *mysql-connector-java*—Native Java driver for MariaDB (optional package).
- *perl-DBD-MySQL*—A MariaDB interface for Perl (optional package).

The **`/etc/my.cnf`** file has default configurations for MariaDB, such as the data directory, socket bindings, and log and error file location.

Note

Instead of adding new configurations to the **/etc/my.cnf** file, a newly created file named ***.cnf** can be added to the **/etc/my.cnf.d/** directory holding the configuration of MariaDB.

MariaDB installation demonstration

In this example, please follow along with these steps while your instructor demonstrates how to install a MariaDB database server.

1. Install MariaDB on server1 with the **yum** command.

```
[root@server1 ~]# yum groupinstall mariadb mariadb-client -y
```

2. Start the MariaDB service on server1 with the **systemctl** command.

```
[root@server1 ~]# systemctl start mariadb
```

Note

The default MariaDB log file is **/var/log/mariadb/mariadb.log**. This file should be the first place to look when troubleshooting MariaDB.

3. Enable the MariaDB service to start at boot on server1.

```
[root@server1 ~]# systemctl enable mariadb
```

4. Verify the status of the service on server1.

```
[root@server1 ~]# systemctl status mariadb
```

The **status** option reports some attributes, if the database is started:

- Loaded—Shows if this service is loaded and enabled.
- Active—Shows if this service is activated.
- Main PID—Shows the main process ID from this service.
- CGroup—Shows all processes that belong to this service.

Note

If the database is stopped, the **status** option will report the last known PID, and that the service is inactive.

Improve MariaDB installation security

MariaDB provides a program to improve security from the baseline install state. Run **`mysql_secure_installation`** without arguments:

```
[root@server1 ~]# mysql_secure_installation
```

This program enables improvement of MariaDB security in the following ways:

- Sets a password for root accounts.
- Removes root accounts that are accessible from outside the local host.
- Removes anonymous-user accounts.
- Removes the test database.

The script is fully interactive, and will prompt for each step in the process.

MariaDB and networking

MariaDB can be configured to be accessed remotely, or limited to just local connections.

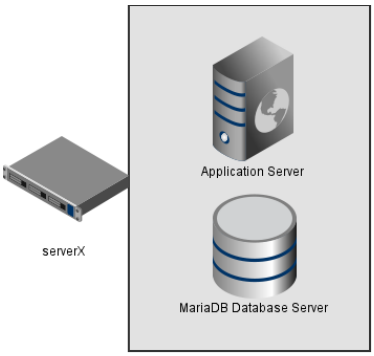

Figure 8.1: Local access to MariaDB

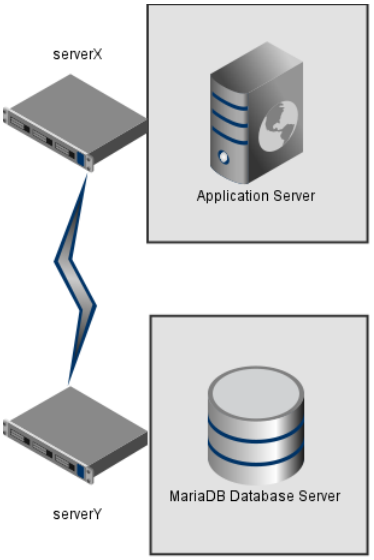

Figure 8.2: Remote access to MariaDB

In the first scenario, the database can only be accessed locally. Security is greatly improved, because the records will only be accessed from applications that are on the same server. The disadvantage is that the server will share the same resources with other services, and this may impact performance in the database server.

In the second scenario, the database can be accessed remotely. In this case, safety decreases because another port is opened on the server, which may result in an attack. On the other hand, the performance of the server increases by not having to share resources.

When MariaDB is accessed remotely, by default, the server listens for TCP/IP connections on all available interfaces on port 3306.

Note

Although the MariaDB service listens on all interfaces by default, no users have remote access permission, also by default.

Configuring MariaDB networking

MariaDB network configuration directives are found in the **`/etc/my.cnf`** file, under the **`[mysqld]`** section.

`bind-address`

The server will listen based on this directive. Only one value may be entered. Possible values are:

- Host name
- IPv4 address
- IPv6 address
- Set this value to **`::`** to connect to all available addresses (IPv6 and IPv4), or leave blank (or set to 0.0.0.0) for all IPv4 addresses.

Important

There can be only one **`bind-address`** entry in **`/etc/my.cnf`**. On a system with multiple addresses, selecting a single address is possible, or all addresses, but nothing in between.

`skip-networking`

If set to 1, the server will listen only for local clients. All interaction with the server will be through a socket, located by default at **`/var/lib/mysql/mysql.sock`**. This location can be changed with a **`socket`** value in **`/etc/my.cnf`**.

Important

Be aware that if networking is shut off in this manner, this disables connections via localhost as well. The MySQL client can still make local connections through the socket file automatically.

`port`
Port to listen on for TCP/IP connections.

Important

For remote access, the firewall needs to be modified. Fortunately, it is a known service, so it can be simply added via:

```
[root@server1 ~]# firewall-cmd --permanent --add-service=mysql
[root@server1 ~]# firewall-cmd --reload
```

References

`mysql_secure_installation`(1), **`mysql.server`**(1), and **`mysqld_selinux`**(8) man pages

> Red Hat Software Collections documentation
> https://access.redhat.com/site/documentation/en-US/Red_Hat_Software_Collections/

Practice: Installing MariaDB

Guided exercise

In this lab, you will install a MariaDB database server.

Resources:	
Machines:	desktop1 and server1

Outcomes:
A MariaDB database server running on your server1 machine.

Before you begin...
- Reset your **server1** machine.

You have been asked to install a MariaDB database server on your server1 machine. You need to secure the MariaDB service, and must configure it to accept connections only from local clients.

1. Install the *mariadb* and *mariadb-client* groups.
   -
```
[root@server1 ~]# yum groupinstall mariadb mariadb-client -y
```

2. Start and enable the **mariadb** service.
   2.1.
```
[root@server1 ~]# systemctl start mariadb
```
   2.2.
```
[root@server1 ~]# systemctl enable mariadb
```

3. Verify that MariaDB is listening on all interfaces.
   -
```
[root@server1 ~]# ss -tulpn | grep mysql

tcp  LISTEN 0  50  *:3306    *:*  users:(("mysqld", 13611,13))
```

Note

*Why grep on **mysql** when you are running MariaDB?* MariaDB is a fork of MySQL which is meant to act as a drop-in replacement.

4. Enable the **skip-networking** directive.
   - Open the file **/etc/my.cnf** in a text editor, and in section [mysqld], add the line
```
skip-networking=1
```

5. Restart the **mariadb** service.

- ```
  [root@server1 ~]# systemctl restart mariadb
  ```

6. Verify that MariaDB is not listening on all interfaces.

   - ```
     [root@server1 ~]# ss -tulpn | grep mysql
     ```

     This command should now return nothing.

7. Secure the Mariadb service using the **mysql_secure_installation** tool. Set the **root** password to **redhat**, and answer *Yes* to all other questions.

   7.1.
   ```
   [root@server1 ~]# mysql_secure_installation

   In order to log into MariaDB to secure it, we'll need the current
   password for the root user.  If you've just installed MariaDB, and
   you haven't set the root password yet, the password will be blank,
   so you should just press enter here.

   Enter current password for root (enter for none):
   OK, successfully used password, moving on...

   Setting the root password ensures that nobody can log into the MariaDB
   root user without the proper authorisation.

   Set root password? [Y/n] Y
   New password: redhat
   Re-enter new password: redhat
   Password updated successfully!
   Reloading privilege tables..
    ... Success!

   By default, a MariaDB installation has an anonymous user, allowing anyone
   to log into MariaDB without having to have a user account created for
   them.  This is intended only for testing, and to make the installation
   go a bit smoother.  You should remove them before moving into a
   production environment.

   Remove anonymous users? [Y/n] Y
    ... Success!

   Normally, root should only be allowed to connect from 'localhost'.  This
   ensures that someone cannot guess at the root password from the network.

   Disallow root login remotely? [Y/n] Y
    ... Success!

   By default, MariaDB comes with a database named 'test' that anyone can
   access.  This is also intended only for testing, and should be removed
   before moving into a production environment.

   Remove test database and access to it? [Y/n] Y
    - Dropping test database...
    ... Success!
    - Removing privileges on test database...
    ... Success!

   Reloading the privilege tables will ensure that all changes made so far
   will take effect immediately.
   ```

```
Reload privilege tables now? [Y/n] Y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

7.2. Verify that the **root** user cannot log in without a password.

```
[root@server1 ~]# mysql -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password:
 NO)
```

7.3. Verify that the **test** database is removed.

```
[root@server1 ~]# mysql -u root -p
Enter password: redhat
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)
MariaDB [(none)]> exit;
Bye
```

Working with MariaDB Databases

Objectives

After completing this section, students should be able to examine, search, create, and change database information using Structured Query Language (SQL) and MariaDB statements.

Creating a database

The installation of the *mariadb-client* group provides a program called **mysql**. With this program, it is possible to connect to a local or remote MariaDB database server.

```
[root@server1 ~]# mysql(1) -u(2) root(3) -h(4) localhost(5) -p(6)
```

1. Client to connect to the MariaDB database server.
2. Option to specify the username for this connection.
3. Username for this connection.
4. Option to specify the host name for this connection. If not specified, the default value is localhost.
5. Host name for this connection.
6. Option to prompt for password.

A database in MariaDB is implemented as a directory. The default installation has four databases. To list the databases, run the command:

```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
4 rows in set (0.01 sec)
```

Important

The only default database that can be erased is **test**.

Note

Unlike the shell, MariaDB—like most relational database systems—is not case-sensitive for simple commands. **`show databases;`** and **`SHOW DATABASES;`** work just as well as **`ShOw DATAbases;`**. However, table and database names *are* case-sensitive. Many databases are set up using all lowercase for the database names, so common practice is to use uppercase for the commands to differentiate the command itself from the target of the command (more examples follow). The important part of these commands, and of most commands entered at the prompt, is to terminate the command with a **`;`**.

To create a new database, run the command:

```
MariaDB [(none)]> CREATE DATABASE inventory;
```

After the creation of the new database, the next step is to connect to this database so that it can be populated with tables and data:

```
MariaDB [(none)]> USE inventory;
```

Note

It is possible to switch between databases at any time with this command.

MariaDB (like all relational database systems) can have multiple tables per database. List the tables with the **`SHOW TABLES;`** command:

```
MariaDB [(none)]>  USE mysql;
MariaDB [(none)]>  SHOW TABLES;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
| general_log               |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| ndb_binlog_index          |
| plugin                    |
| proc                      |
| procs_priv                |
| proxies_priv              |
| servers                   |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
```

```
| user                       |
+----------------------------+
24 rows in set (0.00 sec)
```

To list attributes (or the column names) from a table, use:

```
MariaDB [(mysql)]>  DESCRIBE servers;
+-------------+----------+------+-----+---------+-------+
| Field       | Type     | Null | Key | Default | Extra |
+-------------+----------+------+-----+---------+-------+
| Server_name | char(64) | NO   | PRI |         |       |
| Host        | char(64) | NO   |     |         |       |
| Db          | char(64) | NO   |     |         |       |
| Username    | char(64) | NO   |     |         |       |
| Password    | char(64) | NO   |     |         |       |
| Port        | int(4)   | NO   |     | 0       |       |
| Socket      | char(64) | NO   |     |         |       |
| Wrapper     | char(64) | NO   |     |         |       |
| Owner       | char(64) | NO   |     |         |       |
+-------------+----------+------+-----+---------+-------+
42 rows in set (0.00 sec)
```

This output completely describes the data in the **servers** table in the **mysql** database. The **Port** attribute is stored as an **int**eger, using a maximum of **4** digits, and defaults to **0**.

The **Key** value is null for most of these attributes. Only the **Server_name** has a value: PRI. This sets the attribute as the primary key for the table. Primary keys are unique identifiers for the data in the table. No two entries can have the same primary key, and only one attribute may be set as the primary key. Primary keys are often used to link tables together, and are an important concept when designing complex databases. There are also secondary keys, and composite keys (where multiple attributes together form the unique key). A deeper discussion of keys is beyond the scope of this course.

The **Extra** value is used to show any additional features of the attribute. This value can be complex, but a common one is **auto_increment**, which states that the value of this column will be incremented by 1 for each new entry made into the table. It is a common value for primary keys to have, as seen in later examples.

Using SQL: Structured Query Language

Structured Query Language (SQL) is a special programming language designed for managing data held in relational databases. Some common SQL commands include **insert**, **update**, **delete**, and **select**.

Note

These four basic commands are often referred to by the generic term "CRUD operations." CRUD stands for Create (**insert**), Read (**select**), Update (**update**), and Delete (**delete**).

To **insert** data into a table, the first step is to figure out the attributes of the table.

```
MariaDB [(inventory)]>  DESCRIBE product;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
```

```
+-----------------+-------------+------+-----+---------+----------------+
| id              | int(11)     | NO   | PRI | NULL    | auto_increment |
| name            | varchar(100)| NO   |     | NULL    |                |
| price           | double      | NO   |     | NULL    |                |
| stock           | int(11)     | NO   |     | NULL    |                |
| id_category     | int(11)     | NO   |     | NULL    |                |
| id_manufacturer | int(11)     | NO   |     | NULL    |                |
+-----------------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

In this example, all attributes are required. To insert a new product, the command will be rather long and complicated:

```
MariaDB [(inventory)]> INSERT INTO
 product(1)(name,price,stock,id_category,id_manufacturer)(2) VALUES ('SDSSDP-128G-G25
 2.5',82.04,30,3,1)(3);
Query OK, 1 row affected (0.00 sec)
```

1. Table name.
2. Attributes (columns) that will be inserted.
3. Values that will be inserted in the same order defined by attributes, using the proper data types.

Note

Note that the attribute ID was not specified, even though it is required. When inserting a new record, MariaDB will automatically assign a sequential value for that column. This is because this column is marked as **`auto_increment`**.

Delete a record with the **`delete`** statement:

```
MariaDB [(inventory)]> DELETE FROM product(1) WHERE(2) id = 1(3) ;
Query OK, 1 row affected (0.01 sec)
```

1. Table name.
2. Clause that imposes a condition on the command execution.
3. The condition for the record to be deleted; often the primary key-value pair is used.

Warning

If the **`where`** clause is not specified, *all* records in the table will be erased. This is the database equivalent of running **`rm -rf /`**.

To update a record, use an **`update`** statement:

```
MariaDB [(inventory)]> UPDATE product(1) SET(2) price=89.90, stock=60 WHERE(3) id = 5(4) ;
 Query OK, 1 row affected (0.01 sec)
```

1. Table name.

2. Define the new value from specified attributes.
2. Clause that imposes a condition on the command execution.
3. The condition for the record to be updated.

Warning

If the **where** clause is not specified, all records will be updated.

To read data records from the database, use the **select** statement:

```
MariaDB [(inventory)]>  SELECT name,price,stock(1) FROM product(2);
+---------------------+--------+-------+
| name                | price  | stock |
+---------------------+--------+-------+
| ThinkServer TS140   | 539.88 |    20 |
| RT-AC68U            | 219.99 |    10 |
| X110 64GB           |  73.84 |   100 |
| SDSSDP-128G-G25 2.5 |  82.04 |    30 |
+---------------------+--------+-------+
4 rows in set (0.00 sec)
```

1. Attributes that will be selected.
2. Table name.

To select all attributes, use the wild card *****:

```
MariaDB [(inventory)]>  SELECT * FROM product;
+----+---------------------+--------+-------+-------------+-----------------+
| id | name                | price  | stock | id_category | id_manufacturer |
+----+---------------------+--------+-------+-------------+-----------------+
|  2 | ThinkServer TS140   | 539.88 |    20 |           2 |               4 |
|  3 | RT-AC68U            | 219.99 |    10 |           1 |               3 |
|  4 | X110 64GB           |  73.84 |   100 |           3 |               1 |
|  5 | SDSSDP-128G-G25 2.5 |  82.04 |    30 |           3 |               1 |
+----+---------------------+--------+-------+-------------+-----------------+
4 rows in set (0.00 sec)
```

Filter results with the **where** clause:

```
MariaDB [(inventory)]>  SELECT * FROM product WHERE price > 100;
+----+------------------+--------+-------+-------------+-----------------+
| id | name             | price  | stock | id_category | id_manufacturer |
+----+------------------+--------+-------+-------------+-----------------+
|  2 | ThinkServer TS140 | 539.88 |    20 |           2 |               4 |
|  3 | RT-AC68U          | 219.99 |    10 |           1 |               3 |
+----+------------------+--------+-------+-------------+-----------------+
2 rows in set (0.00 sec)
```

Common Operators for **where** Clauses

Operator	Description
=	Equal
<>	Not equal. Note: In some versions of SQL, this operator may be written as !=
>	Greater than

Operator	Description
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

References

mysql(1) man page

> MariaDB Knowledge Base: *SQL Commands*
> https://mariadb.com/kb/en/sql-commands/

> MariaDB Knowledge Base: *Data Types*
> https://mariadb.com/kb/en/data-types/

Managing Database Users and Access Rights

Objectives

After completing this section, students should be able to configure MariaDB users and grant or revoke database access rights.

Creating user accounts with MariaDB

By default, MariaDB handles authentication and authorization through the **`user`** table in the **`mysql`** database. This means that the root password for the database is persisted in the **`user`** table and not in the operating system.

Note

Recent versions of MariaDB can use PAM for authentication on Linux.

The **`CREATE USER`** statement creates new accounts. This statement will create a new row in the **`mysql.user`** table that has no privileges.

Note

`information_schema` and **`test`** database allows some privileges for all users. This is the main reason why the **`test`** database is often deleted.

Important

To create a new user, the connected user must have the global CREATE USER privilege or the INSERT privilege for the mysql database.

Account names are specified as **`'user_name'@'host_name'`**. This makes it possible to create multiple user accounts with the same name, but with different privileges according to the source host (that is, the host from which the user is connecting).

```
MariaDB [(none)]>  CREATE USER mobius@localhost(1) IDENTIFIED BY 'redhat'(2);
```

(1) Username/host name for this account

(2) Password for this account

This user account can only connect from **`localhost`**, with the password **`redhat`**, and has no privileges. Passwords are encrypted in the **`user`** table:

```
MariaDB [mysql]> SELECT host,user,password FROM user WHERE user = 'mobius';
+-----------+--------+-------------------------------------------+
| host      | user   | password                                  |
+-----------+--------+-------------------------------------------+
| localhost | mobius | *84BB5DF4823DA319BBF86C99624479A198E6EEE9 |
+-----------+--------+-------------------------------------------+
```

```
1 row in set (0.00 sec)
```

When using this account, before granting any privileges, access will be denied for almost any action:

```
[root@server1 ~]# mysql -u mobius -p
Enter password: redhat
MariaDB [(none)]> create database inventory;
ERROR 1044 (42000): Access denied for user 'mobius'@'localhost' to database 'inventory'
```

Note

If the host name is not provided, it is assumed to be "%". This means that this user can access from any source host.

Account Examples

Account	Description
mobius@'localhost'	User mobius can connect just from localhost.
mobius@'192.168.1.5'	User mobius can connect from 192.168.1.5 host.
mobius@'192.168.1.%'	User mobius can connect from any host that belongs to the network 192.168.1.0.
mobius@'%'	User mobius can connect from any host.
mobius@'2000:472:18:b51:c32:a21'	User mobius can connect from 2000:472:18:b51:c32:a21 host.

Granting and revoking privileges for user accounts

Privileges are the permissions that the user may have within MariaDB. The privileges are organized as:

- Global privileges, such as **`CREATE USER`** and **`SHOW DATABASES`**, for administration of the database server itself.
- Database privileges, such as **`CREATE`** for creating databases and working with databases on the server at a high level.
- Table privileges, such as the CRUD commands, for creating tables and manipulating data in the database.
- Column privileges, for granting table-like command usage, but on a particular column (generally rare).
- Other, more granular privileges, which are discussed in detail in the MariaDB documentation.

The **`GRANT`** statement can be used to grant privileges to accounts. The connected user must have the **`GRANT OPTION`** privilege (a special privilege that exists at several levels) to grant privileges. A user may only grant privileges to others that have already been granted to that user (for example, **`mobius`** cannot grant **`SELECT`** privileges on a database table unless **`mobius`** already has that privilege *and* the **`GRANT OPTION`** table privilege).

```
[root@server1 ~]# mysql -u mobius -p
```

```
Enter password: redhat
MariaDB [(none)]> use inventory;
MariaDB [(inventory)]> select * from category;
ERROR 1142 (42000): SELECT command denied to user 'mobius'@'localhost' for table
 'category'
MariaDB [(inventory)]> exit
[root@server1 ~]# mysql -u root -p
Enter password: redhat
MariaDB [(none)]> use inventory;
MariaDB [(inventory)]> GRANT SELECT, UPDATE, DELETE, INSERT(1) on inventory.category(2) to
 mobius@localhost(3);
Query OK, 0 rows affected (0.00 sec)
MariaDB [(inventory)]> exit
[root@server1 ~]# mysql -u mobius -p
Enter password: redhat
MariaDB [(none)]> use inventory;
MariaDB [(inventory)]> select * from category;
+----+------------+
| id | name       |
+----+------------+
|  1 | Networking |
|  2 | Servers    |
|  3 | Ssd        |
+----+------------+
3 rows in set (0.00 sec)
```

1. Define the privileges to be granted (in this case, the full CRUD capabilities are being granted).
2. Define which table the privilege will be granted for.
3. The user to be granted privileges.

Grant Examples

Grant	Description
GRANT SELECT ON database.table TO username@hostname	Grant select privilege for a specific table in a specific database to a specific user.
GRANT SELECT ON database.* TO username@hostname	Grant select privilege for all tables in a specific database to a specific user.
GRANT SELECT ON *.* TO username@hostname	Grant select privilege for all tables in all databases to a specific user.
GRANT CREATE, ALTER, DROP ON database.* to username@hostname	Grant privilege to create, alter, and drop tables in a specific database to a specific user.
GRANT ALL PRIVILEGES ON *.* to username@hostname	Grant all available privileges for all databases to a specific user, effectively creating a superuser, similar to root.

The **REVOKE** statement allows for revoking privileges from accounts. The connected user must have the **GRANT OPTION** privilege and have the privileges that are being revoked to revoke a privilege.

```
MariaDB [(none)]> REVOKE SELECT, UPDATE, DELETE, INSERT(1) on inventory.category(2) from
 mobius@localhost(3);
```

```
Query OK, 0 rows affected (0.00 sec)
```

1. Define the privileges to be revoked.
2. Define for which table the privilege will be revoked.
3. Privilege revoked from this user.

Important

After granting or revoking a privilege, reload all privileges from the privileges tables in the mysql database.

```
MariaDB [(none)]> FLUSH PRIVILEGES;
```

In order to revoke privileges, the list of privileges granted to a user will be needed. The simple command **`SHOW GRANTS FOR username;`** will provide the list of privileges for that user:

```
MariaDB [(none)]> SHOW GRANTS FOR root@localhost;
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
| GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION        |
+---------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

When a user is no longer required, it can be deleted from the database using **`DROP USER username;`**. The **`username`** should use the same **`'user'@'host'`** format used for **`CREATE USER`**.

Important

If an account that is currently connected is **`DROP`**ped, it will not be deleted until the connection is closed. The connection will not be automatically closed.

Troubleshooting database access

Some Common DB Access Issues

Issue	Solution
User has been granted access to connect from any host, but can only connect on localhost using the **`mysql`** command (applications he or she uses cannot connect, even on localhost).	Remove the **`skip-networking`** directive from **`my.cnf`** and restart the service.
User can connect with any application on localhost, but not remotely.	Check the **`bind-address`** configuration in **`my.cnf`** to ensure the database is accessible. Ensure that the **`user`** table includes an entry for the user from the host he is trying to connect from.

Issue	Solution
User can connect, but cannot see any database other than **`information_schema`** and **`test`**.	Common problem when a user has just been created, as they have no privileges by default, though they can connect and use those default databases. Add grants for the database the user requires.
User can connect, but cannot create any databases.	Grant the user the global CREATE privilege (this also grants DROP privileges).
User can connect, but cannot read or write any data.	Grant the user the CRUD privileges for only the database he or she will be using.

References

`mysql`(1) man page

> MariaDB Knowledge Base: *User Account Management*
> https://mariadb.com/kb/en/user-account-management/

> MariaDB Knowledge Base: *Account Management SQL Commands: REVOKE*
> https://mariadb.com/kb/en/revoke/

> MariaDB Knowledge Base: *Account Management SQL Commands: GRANT*, "Database Privileges"
> https://mariadb.com/kb/en/grant/#database-privileges

Practice: Managing Users

Guided exercise

In this lab, you will install a MariaDB server.

Resources:	
Machines:	desktop1 and server1

Outcomes:
One MariaDB user with select privilege and another user with insert, update, delete, and select privileges on your server1 machine.

Before you begin...
- Do not reset your server1 system.
- Log into and set up your server1 system as root.
- Log into and set up your desktop1 system as root.

You have been asked to create two MariaDB users on your server1 machine, according to the following requirements:

User	Accepts connection from host	Password	Privileges
john	localhost	john_password	Insert, update, delete, select on all tables from inventory database
steve	any host	steve_password	Select on all tables from inventory database

1. Set up MariaDB with user root.

   1.1. Modify /etc/my.cnf and comment out **`skip-networking=1`** then restart the service.

```
[root@server1 ~]# vi /etc/my.cnf
[root@server1 ~]# systemctl restart mariadb
```

   1.2. Open the firewall

```
[root@server1 ~]# firewall-cmd --permanent --add-service=mysql
[root@server1 ~]# firewall-cmd --reload
```

   1.3. Connect to MariaDB and create a database:

```
[root@server1 ~]# mysql -u root -p
Password: redhat
MariaDB [(none)]> CREATE DATABASE `inventory`;
USE inventory;
CREATE TABLE `category` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(30) DEFAULT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
LOCK TABLES `category` WRITE;
INSERT INTO `category` VALUES (1,'Networking'),(2,'Servers'),(3,'Ssd');
UNLOCK TABLES;
```

2. Create the users john and steve. Note the passwords are the value of "identified by".

- ```
  MariaDB [(none)]> CREATE USER john@localhost identified by 'john_password';
  MariaDB [(none)]> CREATE USER steve@'%' identified by 'steve_password';
  ```

3. Grant insert, update, delete, and select privileges to user john.

- ```
  MariaDB [(none)]> GRANT INSERT, UPDATE, DELETE, SELECT on inventory.* to
   john@localhost;
  ```

4. Grant select privilege to user steve.

- ```
  MariaDB [(none)]> GRANT SELECT on inventory.* to steve@'%';
  ```

5. Flush the privileges.

- ```
  MariaDB [(none)]> FLUSH PRIVILEGES;
  MariaDB [(none)]> exit;
  ```

6. Connect with user john and verify his privileges.

   6.1. Connect to MariaDB.

   ```
   [root@server1 ~]# mysql -u john -p
   ```

   6.2. Select the inventory database.

   ```
   MariaDB [(none)]> USE inventory;
   ```

   6.3. Verify the select privilege.

   ```
   MariaDB [(inventory)]> SELECT * FROM category;
   +----+------------+
   | id | name       |
   +----+------------+
   |  1 | Networking |
   |  2 | Servers    |
   |  3 | Ssd        |
   +----+------------+
   3 rows in set (0.00 sec)
   ```

   6.4. Verify the insert privilege.

   ```
   MariaDB [(inventory)]> INSERT INTO category(name) VALUES('Memory');
   ```

```
Query OK, 1 row affected (0.00 sec)
```

6.5. Verify the update privilege.

```
MariaDB [(inventory)]> UPDATE category SET name='Solid State Drive' where id =
 3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

6.6. Verify the delete privilege.

```
MariaDB [(inventory)]> DELETE FROM category WHERE name LIKE 'Memory';
Query OK, 1 row affected (0.01 sec)
```

7. On your desktop1 system, connect with user steve and verify his privileges.

7.1. Make sure the MariaDB client is installed.

```
[root@desktop1 ~]# yum -y install mariadb
```

7.2. Connect to MariaDB.

```
[root@desktop1 ~]# mysql -u steve -h server1 -p
```

7.3. Select the inventory database.

```
MariaDB [(none)]> USE inventory;
```

7.4. Verify the select privilege.

```
MariaDB [(inventory)]> SELECT * FROM category;
+----+------------------+
| id | name             |
+----+------------------+
|  1 | Networking       |
|  2 | Servers          |
|  3 | Sold State Drive |
+----+------------------+
3 rows in set (0.00 sec)
```

7.5. Verify the insert privilege.

```
MariaDB [(inventory)]> INSERT INTO category(name) VALUES('Memory');
ERROR 1142 (42000): INSERT command denied to user 'steve'@'desktop1.example.com'
 for table 'category'
```

Creating and Restoring MariaDB Backups

Objectives

After completing this section, students should be able to back up a MariaDB database safely and restore a database backup.

Creating a backup

It is very important to back up MariaDB databases, and databases in general. The database often contains most of a company's mission-critical data (sales, clients, etc.). Performing backups enables a system administrator to recover data after several types of events:

- Operating system crash
- Power failure
- File system crash
- Hardware problem
- Security breach
- Database corruption
- Data poisoning

There are two ways to back up MariaDB:

- Logical
- Physical (raw)

Logical backups export information and records in plain text files, while *physical backups* consist of copies of files and directories that store content.

Logical backups have these characteristics:

- The database structure is retrieved by querying the database.
- Logical backups are highly portable, and can be restored to another database provider (such as Postgres) in some cases.
- Backup is slower because the server must access database information and convert it to a logical format.
- Performed while the server is online.
- Backups do not include log or configuration files.

Physical backups have these characteristics:

- Consist of raw copies of database directories and folders.
- Output is more compact.
- Backups can include log and configuration files.
- Portable only to other machines with similar hardware and software.

- Faster than logical backup.
- Should be performed while the server is offline, or while all tables in the database are locked, preventing changes during the backup.

Performing a logical backup

A logical backup can be done with the **mysqldump** command:

```
[root@server1 ~]# mysqldump -u root(1) -p(2) inventory(3) > /backup/inventory.dump(4)
```

1. User name to connect to MariaDB for backup
2. Prompt for password for this user
3. Selected database for backup
4. Backup file

Note

To logically back up all databases, use option **--all-databases**:

```
[root@server1 ~]# mysqldump -u root -p --all-databases > /backup/mariadb.dump
```

A dump of this kind will include the **mysql** database, which includes all **user** information.

The output of a logical backup will appear to be a series of SQL statements. As an example, here is a snippet from a dump of the **mysql** database:

```
LOCK TABLES `user` WRITE;
/*!40000 ALTER TABLE `user` DISABLE KEYS */;
INSERT INTO `user` VALUES ('localhost','root','','Y','Y','Y','Y','Y','Y','Y','Y'
,'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y'
,'Y','','','','',0,0,0,0,'',''),('localhost.localdomain','root','','Y','Y','Y','
Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','
Y','Y','Y','Y','Y','Y','','','','',0,0,0,0,'',''),('127.0.0.1','root','','Y','Y'
,'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y'
,'Y','Y','Y','Y','Y','Y','Y','Y','','','','',0,0,0,0,'',''),('::1','root','','Y','Y'
,'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y'
,'Y','Y','Y','Y','Y','Y','Y','Y','','','','',0,0,0,0,'',''),('localhost','','','N','
N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','
N','N','N','N','N','N','N','N','','','','',0,0,0,0,'',''),('localhost.localdomai
n','','','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N'
,'N','N','N','N','N','N','N','N','N','N','N','N','','','','',0,0,0,0,'',''),('localh
ost','mobius','*84BB5DF4823DA319BBF86C99624479A198E6EEE9','N','N','N','N','N','N
','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N','N
','N','N','N','','','','',0,0,0,0,'','');
/*!40000 ALTER TABLE `user` ENABLE KEYS */;
UNLOCK TABLES;
```

Notice the encrypted password for **mobius** is easily visible, so take care with storage of backups of this kind. Also, individual tables are locked and unlocked by default as they are read from during a logical backup.

Important

mysqldump requires at least the **Select** privilege for dumped tables, **SHOW VIEW** for dumped views, and **TRIGGER** for dumped triggers.

Useful Options

Option	Description
--add-drop-tables	Tells MariaDB to add a DROP TABLE statement before each CREATE TABLE statement.
--no-data	Dumps only the database structure, not the contents.
--lock-all-tables	No new record can be inserted anywhere in the database while the copy is finished. This option is very important to ensure backup integrity.
--add-drop-databases	Tells MariaDB to add a DROP DATABASE statement before each CREATE DATABASE statement.

Performing a physical backup

Several tools are available to perform physical backups, such as **ibbackup**, **cp**, **mysqlhotcopy**, and **lvm**.

A MariaDB physical backup task can use the known benefits of LVM snapshots. The following process will back up MariaDB using LVM.

Important

The key benefit of this methodology is that it is very quick, and keeps the downtime of the database short. This is a great argument for putting the database files on a dedicated LVM partition.

Verify where MariaDB files are stored:

```
[root@server1 ~]# mysqladmin variables | grep datadir
| datadir                         | /var/lib/mysql/             |
```

Verify which logical volume hosts this location:

```
[root@server1 ~]# df /var/lib/mysql
Filesystem              1K-blocks    Used Available Use% Mounted on
/dev/mapper/vg0-mariadb  51475068 7320316  41516928  15% /var/lib/mysql            |
```

This shows that the volume group is **vg0** and the logical volume name is **mariadb**.

Verify how much space is available for the snapshot:

```
[root@server1 ~]# vgdisplay vg0 | grep Free
```

```
Free  PE / Size       15321 / 61.29 GB           |
```

This shows that 61.29 GB are available for a snapshot.

Connect to MariaDB, flush the tables to disk, and lock them (alternately, shut down the **mariadb** service):

Warning

This step is very important so that no new records are inserted into the database while the snapshot is created.

```
[root@server1 ~]# mysql -u root -p
MariaDB [(none)]> FLUSH TABLES WITH READ LOCK;
```

Warning

Do not close this session. As soon as the client disconnects, this lock is lifted. The database *must* remained locked until the LVM snapshot is created.

In another terminal session, create the LVM snapshot:

```
[root@server1 ~]# lvcreate -L20G -s -n mariadb-backup /dev/vg0/mariadb           |
```

Important

This snapshot needs to be large enough to hold the backup.

In the original MariaDB session, unlock the tables (or, bring the **mariadb** service up):

```
MariaDB [(none)]> UNLOCK TABLES;
```

The snapshot can now be mounted at an arbitrary location:

```
[root@server1 ~]# mkdir /mnt/snapshot
[root@server1 ~]# mount /dev/vg0/mariadb-backup /mnt/snapshot
```

From here, any standard file system backup can be used to store a copy of **/var/lib/mysql** as mounted under **/mnt/snapshot**.

Important

Do not forget to delete the snapshot once it has been backed up.

```
[root@server1 ~]# umount /mnt/snapshot
[root@server1 ~]# lvremove /dev/vg0/mariadb-backup
```

Restoring a backup

Logical restore
A logical restore can be done with the command **mysql**:

```
[root@server1 ~]# mysql -u root❶ -p ❷ inventory ❸ < /backup/mariadb.dump ❹
```

❶ User to connect with to restore the MariaDB backup (generally root or some other superuser)
❷ Password for this user
❸ Selected database for restore backup
❹ Backup file

Physical restore
To do a physical restore, the mariadb service must be stopped:

```
[root@server1 ~]#  systemctl stop mariadb
```

Verify where MariaDB files are stored:

```
[root@server1 ~]# mysqladmin variables | grep datadir
| datadir                         | /var/lib/mysql/             |
```

Remove the actual content:

```
[root@server1 ~]# rm -rf /var/lib/mysql/*
```

From here, any standard file system restore can be used to restore a copy from backup to **/var/lib/mysql**.

References

mysql(1), **mysqladmin**(1), **mysqldump**(1), **lvcreate**(8), **lvremove**(1), and **vgdisplay**(8) man pages

> MariaDB Knowledge Base: *Backup and Restore Overview*
> https://mariadb.com/kb/en/backup-and-restore-overview/

Practice: Restoring a MariaDB Database from Backup

Guided exercise

In this lab, you will restore a database from a MariaDB logical backup.

Resources:	
Files:	`/root/inventory.dump`
Machines:	server1

Outcomes:
A MariaDB database with restored data records.

Before you begin...

- Do not reset your server1 system.
- Log into and set up your server1 system as root.

Backup, destroy, and then restore from a logical backup.

1. Backup the inventory database to a file:

   ```
   [root@server1 ~]# mysqldump -p inventory > /root/inventory.dump
   ```

2. Connect to MariaDB as user root and drop old database.

   ```
   [root@server1 ~]# mysql -u root -p
   MariaDB [(none)]> drop database inventory;
   MariaDB [(none)]> use inventory;
   ERROR 1049 (42000): Unknown database 'inventory'
   ```

3. Create a new database called inventory.

   ```
   MariaDB [(none)]> create database inventory;
   MariaDB [(none)]> exit;
   ```

4. Restore from the logical backup.

   ```
   [root@server1 ~]# mysql -u root -p inventory < /root/inventory.dump
   ```

5. Verify restored data.

   5.1. Connect again to MariaDB.

   ```
   [root@server1 ~]# mysql -u root -p
   ```

5.2. Connect to the inventory database.

```
MariaDB [(none)]> use inventory;
```

5.3. Select all categories.

```
MariaDB [(inventory)]> SELECT * FROM category;
+----+-------------------+
| id | name              |
+----+-------------------+
|  1 | Networking        |
|  2 | Servers           |
|  3 | Solid State Drive |
+----+-------------------+
3 rows in set (0.00 sec)
```

Summary

Installing MariaDB

In this section, students learned how to:

- Describe relational databases.
- Describe database concepts.
- Perform a MariaDB installation.

Working with MariaDB Databases

In this section, students learned how to:

- Examine databases.
- Search in databases.
- Create databases.
- Use SQL.

Managing Database Users and Access Rights

In this section, students learned how to:

- Create users.
- Grant privileges.
- Revoke privileges.
- Flush privileges.

Creating and Restoring MariaDB Backups

In this section, students learned how to:

- Back up a MariaDB database.
- Restore a MariaDB database.

CHAPTER 9

PROVIDING APACHE HTTPD WEB SERVICE

Overview	
Goal	To configure Apache HTTPD to provide TLS-enabled websites and virtual hosts.
Objectives	• Identify the key configuration files, log files, and content directories used by Apache **`httpd`**. • Configure Apache **`httpd`** to provide IP-based and name-based virtual hosts. • Configure Apache **`httpd`** to provide TLS-encrypted virtual hosts. • Configure Apache **`httpd`** to serve dynamic database-driven web content.
Sections	• Configuring Apache HTTPD (and Practice) • Configuring and Troubleshooting Virtual Hosts (and Practice) • Configuring HTTPS (and Practice) • Integrating Dynamic Web Content (and Practice)
Lab	• Configuring a Web Application

Configuring Apache HTTPD

Objectives

After completing this section, students should be able to identify the key configuration files, logfiles, and content directories used by Apache **`httpd`**.

Introduction to Apache HTTPD

Apache HTTPD is one of the most used web servers on the Internet. A web server is a daemon that speaks the **`http(s)`** protocol, a text-based protocol for sending and receiving objects over a network connection. The **`http`** protocol is sent over the wire in clear text, using port **`80/TCP`** by default (though other ports can be used). There is also a TLS/SSL encrypted version of the protocol called **`https`** that uses port **`443/TCP`** by default.

A basic http exchange has the client connecting to the server, and then requesting a resource using the **`GET`** command. Other commands like **`HEAD`** and **`POST`** exist, allowing clients to request just metadata for a resource, or send the server more information.

The following is an extract from a short http exchange:

```
GET /hello.html HTTP/1.1
Host: webapp0.example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cache-Control: max-age=0
```

The client starts by requesting a resource (the **`GET`** command), and then follows up with some extra headers, telling the server what types of encoding it can accept, what language it would prefer, etc. The request is ended with an empty line.

```
HTTP/1.1 200 OK
Date: Tue, 27 May 2014 09:57:40 GMT
Server: Apache/2.4.6 (Red Hat) OpenSSL/1.0.1e-fips mod_wsgi/3.4 Python/2.7.5
Content-Length: 12
Keep-Alive: timeout=5, max=82
Connection: Keep-Alive
Content-Type: text/plain; charset=UTF-8

Hello World!
```

The server then replies with a status code (**`HTTP/1.1 200 OK)`**, followed by a list of headers. The **`Content-Type`** header is a mandatory one, telling the client what type of content is being sent. After the headers are done, the server sends an empty line, followed by the requested content. The length of this content must match the length indicated in the **`Content-Length`** header.

While the http protocol seems easy at first, implementing all of the protocol—along with security measures, support for clients not adhering fully to the standard, and support for dynamically generated pages—is not an easy task. That is why most application developers do not write their

own web servers, but instead write their applications to be run behind a web server like Apache HTTPD.

About Apache HTTPD

Apache HTTPD, sometimes just called “Apache” or **`httpd`**, implements a fully configurable and extendable web server with full **`http`** support. The functionality of **`httpd`** can be extended with *modules*, small pieces of code that plug into the main web server framework and extend its functionality.

On Red Hat Enterprise Linux 7 Apache HTTPD is provided in the *httpd* package. The *web-server* package group will install not only the *httpd* package itself, but also the *httpd-manual* package. Once *httpd-manual* is installed, and the **`httpd.service`** service is started, the *full* Apache HTTPD manual is available on `http://localhost/manual`. This manual has a complete reference of all the configuration directives for **`httpd`**, along with examples. This makes it an invaluable resource while configuring **`httpd`**.

Red Hat Enterprise Linux 7 also ships an environment group called *web-server-environment*. This environment group pulls in the *web-server* group by default, but has a number of other groups, like backup tool and database clients, marked as optional.

A default dependency of the *httpd* package is the *httpd-tools* package. This package includes tools to manipulate password maps and databases, tools to resolve IP addresses in logfiles to hostnames, and a tool (**`ab`**) to benchmark and stress-test web servers.

Basic Apache HTTPD configuration

After installing the *web-server* package group, or the *httpd* package, a default configuration is written to **`/etc/httpd/conf/httpd.conf`**.

This configuration serves out the contents of **`/var/www/html`** for requests coming in to any hostname over plain **`http`**.

The basic syntax of the **`httpd.conf`** is comprised of two parts: **`Key Value`** configuration directives, and HTML-like **`<Blockname parameter>`** blocks with other configuration directives embedded in them. Key/value pairs outside of a block affect the entire server configuration, while directives inside a block typically only apply to a part of the configuration indicated by the block, or when the requirement set by the block is met.

```
ServerRoot "/etc/httpd" (1)
Listen 80 (2)
Include conf.modules.d/*.conf (3)
User apache (4)
Group apache (5)
ServerAdmin root@localhost (6)
<Directory /> (7)
    AllowOverride none
    Require all denied
</Directory>
DocumentRoot "/var/www/html" (8)
<Directory "/var/www">
    AllowOverride None
    Require all granted
</Directory>
```

```
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

<IfModule dir_module>(9)
    DirectoryIndex index.html
</IfModule>

<Files ".ht*">(10)
    Require all denied
</Files>

ErrorLog "logs/error_log"(11)
LogLevel warn
<IfModule log_config_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
    <IfModule logio_module>
      LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O"
 combinedio
    </IfModule>

    CustomLog "logs/access_log" combined(12)
</IfModule>

AddDefaultCharset UTF-8(13)

IncludeOptional conf.d/*.conf(14)
```

(1) This directive specifies where **httpd** will look for any files referenced in the configuration files with a relative path name.

(2) This directive tells **httpd** to start listening on port **80/TCP** on all interfaces. To only listen on select interfaces, the syntax "**Listen 1.2.3.4:80**" can be used for IPv4 or "**Listen [2001:db8::1]:80**" for IPv6.

Note: Multiple **listen** directives are allowed, but overlapping **listen** directives will result in a fatal error, preventing **httpd** from starting.

(3) This directive includes other files, as if they were inserted into the configuration file in place of the **Include** statement. When multiple files are specified, they will be sorted by filename in alphanumeric order before being included. Filenames can either be absolute, or relative to **ServerRoot**, and include wildcards such as *****.

Note: Specifying a nonexistent file will result in a fatal error, preventing **httpd** from starting.

(4)(5) These two directives specify the user and group the **httpd** daemon should run as. **httpd** is always started as **root**, but once all actions that need **root** privileges have been performed —for example, binding to a port number under **1024**—privileges will be dropped and execution is continued as a nonprivileged user. This is a security measure.

(6) Some error pages generated by **httpd** can include a link where users can report a problem. Setting this directive to a valid email address will make the webmaster easier to contact for users. Leaving this setting at the default of **root@localhost** is not recommended.

(7) A **<Directory>** block sets configuration directives for the specified directory, and all descendent directories.

Common directives inside a **<Directory>** block include the following:

- **AllowOverride None**: **.htaccess** files will not be consulted for per-directory configuration settings. Setting this to any other setting will have a performance penalty, as well as the possible security ramifications.

- **`Require All Denied`**: **`httpd`** will refuse to serve content out of this directory, returning a **`HTTP/1.1 403 Forbidden`** error when requested by a client.
- **`Require All Granted`**: Allow access to this directory. Setting this on a directory outside of the normal content tree can have security implications.
- **`Options [`*`[+|-]OPTIONS`*`]...`**: Turn on (or off) certain options for a directory. For example, the **`Indexes`** option will show a directory listing if a directory is requested and no **`index.html`** file exists in that directory.

8. This setting determines where **`httpd`** will search for requested files. It is important that the directory specified here is both readable by **`httpd`** (both regular permissions and SELinux), and that a corresponding **`<Directory>`** block has been declared to allow access.
9. This block only applies its contents if the specified extension module is loaded. In this case, the **`dir_module`** is loaded, so the **`DirectoryIndex`** directive can be used to specify what file should be used when a directory is requested.
10. A **`<Files>`** block works just as a **`<Directory>`** block, but here options for individual (wildcarded) files is used. In this case, the block prevents **`httpd`** from serving out any security-sensitive files like **`.htaccess`** and **`.htpasswd`**.
11. This specifies to what file **`httpd`** should log any errors it encounters. Since this is a relative pathname, it will be prepended with the **`ServerRoot`** directive. In a default configuration, **`/etc/httpd/logs`** is a symbolic link to **`/var/log/httpd/`**.
12. The **`CustomLog`** directive takes two parameters: a file to log to, and a log format defined with the **`LogFormat`** directive. Using these directives, administrators can log exactly the information they need or want. Most log parsing tools will assume that the default **`combined`** format is used.
13. This setting adds a **`charset`** part to the **`Content-Type`** header for **`text/plain`** and **`text/html`** resources. This can be disabled with **`AddDefaultCharset Off`**.
14. This works the same as a regular include, but if no files are found, no error is generated.

Starting Apache HTTPD

`httpd` can be started from the **`httpd.service`** **`systemd`** unit.

```
[root@server1 ~]# systemctl enable httpd.service
[root@server1 ~]# systemctl start httpd.service
```

Once **`httpd`** is started, status information can be requested with **`systemctl status -l httpd.service`**. If **`httpd`** has failed to start for any reason, this output will typically give a clear indication of *why* **`httpd`** failed to start.

Network security

`firewalld` has two predefined services for **`httpd`**. The **`http`** service opens port **`80/TCP`**, and the **`https`** service opens port **`443/TCP`**.

```
[root@server1 ~]# firewall-cmd --permanent --add-service=http --add-service=https
[root@server1 ~]# firewall-cmd --reload
```

In a default configuration, **`SELinux`** only allows **`httpd`** to bind to a specific set of ports. This full list can be requested with the command **`semanage port -l | grep '^http_'`**. For a full overview of the allowed port contexts, and their intended usage, consult the **`httpd_selinux`**(8) man page from the *selinux-policy-devel* package.

Using an alternate document root

Content does not need to be served out of **/var/www/html**, but when changing the **DocumentRoot** setting, a number of other changes must be made as well:

- File system permissions: Any new **DocumentRoot** must be readable by the **apache** user or the **apache** group. In most cases, the **DocumentRoot** should *never* be writable by the **apache** user or group.
- SELinux: The default SELinux policy is restrictive as to what contexts can be read by **httpd**. The default context for web server content is **httpd_sys_content_t**. Rules are already in place to relabel **/srv/*/www/** with this context as well. To serve content from outside of these standard locations, a new context rule must be added with **semanage**.

```
[root@server1 ~]# semanage fcontext -a -t httpd_sys_content_t '/new/location(/.*)?'
```

Consult the **httpd_selinux**(8) man page from the *selinux-policy-devel* package for additional allowed file contexts and their intended usage.

Allowing write access to a DocumentRoot

In a default configuration only **root** has write access to the standard **DocumentRoot**. To allow web developers to write into the **DocumentRoot**, a number of approaches can be taken.

- Set a (default) ACL for the web developers on the **DocumentRoot**. For example, if all web developers are part of the **webmasters** group, and **/var/www/html** is used as the **DocumentRoot**, the following commands will give them write access:

```
[root@server1 ~]# setfacl -R -m g:webmasters:rwX /var/www/html
[root@server1 ~]# setfacl -R -m d:g:webmasters:rwx /var/www/html
```

Important

The uppercase **X** bit sets the executeable bit only on directories instead of directories and regular files. This is especially relevant when done in conjunction with a recursive action on a directory tree.

- Create the new **DocumentRoot** owned by the **webmasters** group, the sticky bit set.

```
[root@server1 ~]# mkdir -p -m 2775 /new/docroot
[root@server1 ~]# chgrp webmasters /new/docroot
```

- A combination of the previous, with the other permissions closed off, and an ACL added for the **apache** group.

References

`httpd`(8) and **`httpd_selinux`**(8) man pages

httpd-manual package contents

Practice: Configuring a Web Server

Guided exercise

In this lab, you will configure a basic **httpd** web server to serve out a static page from the default location, as well as the Apache **httpd** manual.

Resources:	
Machines:	**desktop1** and **server1**

Outcomes:
An Apache **httpd** web server running on your **server1** machine, serving out a static page and the complete Apache **httpd** manual.

Before you begin...
- Reset your **server1** machine.

You have been asked to configure a basic web server on your **server1** machine. This web server should serve out the text “Hello Class!” when the URL `http://server1.example.com/` is requested.

To aid your end users in submitting bug reports, the default error page should include a **mailto:** reference to the email address **webmaster@server1.example.com**.

Since your organization is planning on further customizing the behavior of this web server, the full Apache **httpd** manual should be available under `http://server1.example.com/manual/`.

1. Begin by installing the *httpd* and *httpd-manual* packages.
   - ```
     [root@server1 ~]# yum -y install httpd httpd-manual
     ```
2. Set the **ServerAdmin** directive for the main site configuration to point to **webmaster@server1.example.com**.
   - Open **/etc/httpd/conf/httpd.conf** in a text editor with **root** privileges, and change the line that starts with **ServerAdmin** to the following:
     ```
     ServerAdmin webmaster@server1.example.com
     ```
3. Create the default content page.
   - Create the **/var/www/html/index.html** file with a text editor as user **root**, and add the following content:
     ```
     Hello Class!
     ```
4. Start and enable the **httpd** service.
   - ```
     [root@server1 ~]# systemctl start httpd.service
     ```

```
[root@server1 ~]# systemctl enable httpd.service
```

5. Open all the relevant ports for **http** on the firewall on **server1**.

- ```
  [root@server1 ~]# firewall-cmd --permanent --add-service=http
  [root@server1 ~]# firewall-cmd --reload
  ```

6. Test if you can access the new static page, as well as the Apache **httpd** manual, from your **desktop1** machine.

   6.1. From **desktop1** use curl to access the default web page.

   ```
   [student@desktop1 ~]$ curl -s http://server1.example.com
   Hello Class!
   ```

   6.2. From **desktop1** use curl to access the manual web page.

   ```
   [student@desktop1 ~]$ curl -Is http://server1.example.com/manual/ | head
   HTTP/1.1 200 OK
   Date: Thu, 04 Dec 2014 19:02:06 GMT
   Server: Apache/2.4.6 (Red Hat)
   Last-Modified: Thu, 20 Mar 2014 11:17:16 GMT
   ETag: "22b1-4f507e9630700"
   Accept-Ranges: bytes
   Content-Length: 8881
   Content-Type: text/html; charset=UTF-8
   ```

Note

Alternatively you can open a web browser on desktop1 to open these URLs.
   ```

Configuring and Troubleshooting Virtual Hosts

Objectives

After completing this section, students should be able to configure Apache **`httpd`** to provide IP-based and name-based virtual hosts.

Virtual hosts

Virtual hosts allow a single **`httpd`** server to serve content for multiple domains. Based on either the IP address of the server that was connected to, the hostname requested by the client in the http request, or a combination of both, **`httpd`** can use different configuration settings, including a different **`DocumentRoot`**.

Virtual hosts are typically used when it is not cost-effective to spin up multiple (virtual) machines to serve out many low-traffic sites; for example, in a shared hosting environment.

Configuring virtual hosts

Virtual hosts are configured using **`<VirtualHost>`** blocks inside the main configuration. To ease administration, these virtual host blocks are typically not defined inside **`/etc/httpd/conf/httpd.conf`**, but rather in separate **`.conf`** files in **`/etc/httpd/conf.d/`**.

The following is an example file, **`/etc/httpd/conf.d/site1.conf`**.

```
<Directory /srv/site1/www>(1)
  Require all granted
  AllowOverride None
</Directory>

<VirtualHost 192.168.0.1:80>(2)
  DocumentRoot /srv/site1/www(3)
  ServerName site1.example.com(4)
  ServerAdmin webmaster@site1.example.com(5)
  ErrorLog "logs/site1_error_log"(6)
  CustomLog "logs/site1_access_log" combined(7)
</VirtualHost>
```

1. This block provides access to the **`DocumentRoot`** defined further down.
2. This is the main tag of the block. The **`192.168.0.1:80`** part indicates to **`httpd`** that this block should be *considered* for all connections coming in on that IP/port combination.
3. Here the **`DocumentRoot`** is being set, but only for within this virtual host.
4. This setting is used to configure *name-based* virtual hosting. If multiple **`<VirtualHost>`** blocks are declared for the same IP/port combination, the block that matches **`ServerName`** with the **`hostname:`** header sent in the client **`http`** request will be used.

   There can be exactly zero or one **`ServerName`** directives inside a single **`<VirtualHost>`** block. If a single virtual host needs to be used for more than one domain name, one or more **`ServerAlias`** statements can be used.

5. To help with sorting mail messages regarding the different websites, it is helpful to set unique **ServerAdmin** addresses for all virtual hosts.
6. The location for all error messages related to this virtual host.
7. The location for all access messages regarding this virtual host.

If a setting is not made explicitly for a virtual host, the same setting from the main configuration will be used.

Name-based vs. IP-based virtual hosting

By default, every virtual host is an IP-based virtual host, sorting traffic to the virtual hosts based on what IP address the client had connected to. If there are multiple virtual hosts declared for a single IP/port combination, the **ServerName** and **ServerAlias** directives will be consulted, effectively enabling name-based virtual hosting.

Wildcards and priority

The IP address part of a **<VirtualHost>** directive can be replaced with one of two wildcards: **_default_** and *****. Both have exactly the same meaning: “Match Anything”.

When a request comes in, **httpd** will first try to match against virtual hosts that have an explicit IP address set. If those matches fail, virtual hosts with a wildcard IP address are inspected. If there is still no match, the “main” server configuration is used.

Important

A **<VirtualHost *:80>** will always match for regular http traffic on port **80/TCP**, effectively disabling the main server configuration from ever being used for traffic on port **80/TCP**.

If no exact match has been found for a **ServerName** or **ServerAlias** directive, and there are multiple virtual hosts defined for the IP/port combination the request came in on, the *first* virtual host that matches an IP/port is used, with *first* being seen as the order in which virtual hosts are defined in the configuration file.

When using multiple ***.conf** files, they will be included in alphanumeric sorting order. To create a catch-all (default) virtual host, the configuration file should be named something like **00-default.conf** to make sure that it is included before any others.

Troubleshooting virtual hosts

When troubleshooting virtual hosts, there are a number of approaches that can help.

- Configure a separate **DocumentRoot** for each virtual host, with identifying content.
- Configure separate logfiles, both for error logging and access logging, for each virtual host.
- Evaluate the order in which the virtual host definitions are parsed by **httpd**. Included files are read in alphanumeric sort order based on their filenames.
- Disable virtual hosts one by one to isolate the problem. Virtual host definitions can be commented out of the configuration file(s), and include files can be temporarily renamed to something that does not end in **.conf**.
- **journalctl UNIT=httpd.service** can isolate log messages from just the **httpd.service** service.

References

`httpd`(8) man page

httpd-manual package contents

Practice: Configuring a Virtual Host

Guided exercise

In this lab, you will configure a new web server to serve out content for multiple virtual hosts.

Resources:	
Machines:	**desktop1** and **server1**

Outcomes:
A new web server running on **server1**, serving out content for `www1.example.com` from **/srv/www1.example.com/www/**, and all other domains from **/srv/default/www/**.

Before you begin...
- Reset your **server1** machine.

Over the past few years, your company has been spinning up many web servers for new projects. Unfortunately, there was no structure or coordination between the various projects.

In an effort to clean up the mess, you have been asked to consolidate these various web servers into one, serving out the different domains using name-based virtual hosting.

For now, you will only have to set up a *default* virtual host that serves out a placeholder site from **/srv/default/www/**, and a virtual host for `www1.example.com` that serves out content from **/srv/www1.example.com/www**.

DNS CNAME records for the relevant domains have already been converted to point at your **server1** machine.

1. Start by installing the *httpd* package.
   -
```
[root@server1 ~]# yum install httpd
```
2. 2.1. Create the directories.

```
[root@server1 ~]# mkdir -p /srv/{default,www1.example.com}/www
```

   2.2. Create the placeholder site index.html:

```
[root@server1 ~]# echo 'Coming Soon!' > /srv/default/www/index.html
```

   2.3. Create the index.html for the `www1.example.com` site:

```
[root@server1 ~]# echo "www1" > /srv/www1.example.com/www/index.html
```

   2.4. Create the **index.html** files using a text editor. **/srv/default/www/index.html** gets the "**Coming Soon!**" text, and the **/srv/www1.example.com/www/index.html** file should read "**www1**".

   2.5. Add a valid httpd content context for the new directory.

```
[root@server1 ~]# semanage fcontext -a -t httpd_sys_content_t '/srv(/.*)?'
```

2.6. Reset the context on the files you just created to match the policy and check to make sure httpd_sys_content_t is set on the directories.

```
[root@server1 ~]# restorecon -Rv /srv/
[root@server1 ~]# ls -Z /srv/
drwxr-xr-x. root root unconfined_u:object_r:httpd_sys_content_t:s0 default
drwxr-xr-x. root root unconfined_u:object_r:httpd_sys_content_t:s0
 www1.example.com
```

3. Create a new virtual host definition for the **_default_:80** virtual host. This virtual host should serve out content from **/srv/default/www/**, and log to **logs/default-vhost.log** using the combined format.

3.1. Create a new file called **/etc/httpd/conf.d/00-default-vhost.conf**. Give it the following content:

```
<VirtualHost _default_:80>
  DocumentRoot /srv/default/www
  CustomLog "logs/default-vhost.log" combined
</VirtualHost>
```

3.2. Since in a default configuration, **httpd** blocks access to all directories, you will need to open up the content directory for your default vhost. Add the following block to **/etc/httpd/conf.d/00-default-vhost.conf**.

```
<Directory /srv/default/www>
  Require all granted
</Directory>
```

4. Create a new virtual host definition for a `www1.example.com` virtual host in **/etc/httpd/conf.d/01-www1.example.com-vhost.conf**. This virtual host should respond to requests for both `www1.example.com` and `www1`, serve out content from **/srv/www1.example.com/www**, and store logs in **logs/www1.example.com.log**.

- Create the file **/etc/httpd/conf.d/01-www1.example.com-vhost.conf** with the following contents:

```
<VirtualHost *:80>
  ServerName www1.example.com
  ServerAlias www1
  DocumentRoot /srv/www1.example.com/www
  CustomLog "logs/www1.example.com.log" combined
</VirtualHost>

<Directory /srv/www1.example.com/www>
  Require all granted
</Directory>
```

5. Start and enable the **httpd** service.

- ```
  [root@server1 ~]# systemctl start httpd.service
  [root@server1 ~]# systemctl enable httpd.service
  ```

6. Open up the firewall on **server1** to allow traffic to the **httpd** service.

   - ```
     [root@server1 ~]# firewall-cmd --permanent --add-service=http
     [root@server1 ~]# firewall-cmd --reload
     ```

7. From your **desktop1** system, use curl to visit the following URLs; the first two should respond with the "**www1**" text, while the last two should respond with "**Coming Soon!**".

   - `http://www1.example.com`

     ```
     [student@desktop1 ~]$ curl -s http://www1.example.com
     www1
     ```

   - `http://www1`

     ```
     [student@desktop1 ~]$ curl -s http://www1
     www1
     ```

   - `http://server1.example.com`

     ```
     [student@desktop1 ~]$ curl -s http://server1.example.com
     Coming Soon!
     ```

   - `http://192.168.0.101`

     ```
     [student@desktop1 ~]$ curl -s http://192.168.0.101
     Coming Soon!
     ```

Note

You can also use a web browser on desktop1 to access these URLs.

Configuring HTTPS

Objectives

After completing this section, students should be able to configure Apache **`httpd`** to provide TLS-encrypted virtual hosts.

Transport Layer Security (TLS)

Transport Layer Security (TLS) is a method for encrypting network communications. TLS is the successor to *Secure Sockets Layer* (SSL). TLS allows a client to verify the identity of the server and, optionally, allows the server to verify the identity of the client.

TLS is based around the concepts of *certificates*. A certificate has multiple parts: a public key, server identity, and a signature from a *certificate authority*. The corresponding private key is never made public. Any data encrypted with the private key can only be decrypted with the public key, and vice versa.

During the initial *handshake*, when setting up the encrypted connection, the client and server agree on a set of encryption ciphers supported by both the server and the client, and they exchange bits of random data. The client uses this random data to generate a *session key*, a key that will be used for much faster *symmetric* encryption, where the same key is used for both encryption and decryption. To make sure that this key is not compromised, it is sent to the server encrypted with the server's public key (part of the server certificate).

The following diagram shows a (simplified) version of a TLS handshake.

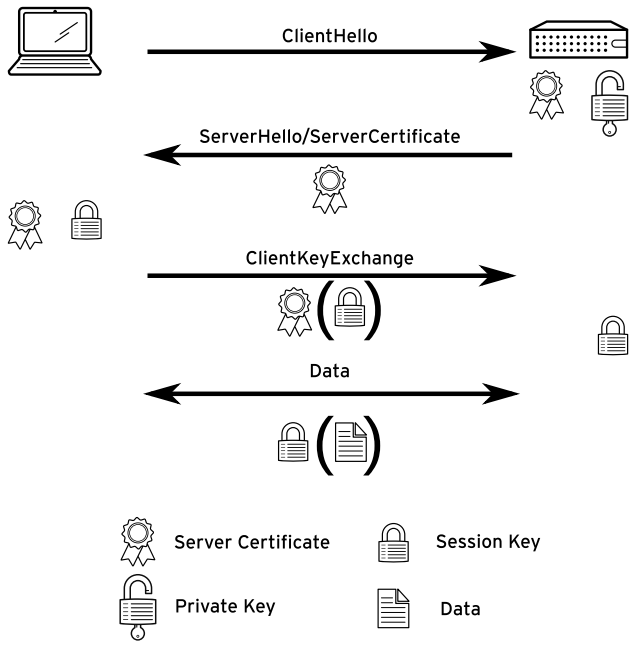

Figure 9.1: A simplified representation of a TLS handshake

1. The client initiates a connection to the server with a **`ClientHello`** message. As part of this message, the client sends a 32-byte random number including a timestamp, and a list of encryption protocols and ciphers supported by the client.

2. The server responds with a **`ServerHello`** message, containing another 32-byte random number with a timestamp, and the encryption protocol and ciphers the client should use.

   The server also sends the server certificate, which consists of a public key, general server identity information like the FQDN, and a signature from a trusted *certificate authority* (CA). This certificate *can* also include the public certificates for all certificate authorities that have signed the certificate, up to a root CA.

3. The client verifies the server certificate by checking if the supplied identity information matches, and by verifying all signatures, checking if they are made by a CA trusted by the client.

   If the certificate verifies, the client creates a *session key* using the random numbers previously exchanged. The client then encrypts this session key using the public key from the server certificate, and sends it to the server using a **`ClientKeyExchange`** message.

4. The server decrypts the session key, and the client and server both start encrypting and decrypting all data send over the connection using the session key.

Note

This is a simplification of the actual protocol; for example, the actual session key never gets transmitted with a lot cipher suites, not even in encrypted form. The server and client both create a *pre-master key* which gets exchanged, and both the server and client calculate the actual session key from that one.

During the negotiations, both the server and client also use a variety of methods to ensure against replay and man-in-the-middle attacks.

Configuring TLS certificates

To configure a virtual host with TLS, multiple steps must be completed:

1. Obtain a (signed) certificate.

2. Install Apache HTTPD extension modules to support TLS.

3. Configure a virtual host to use TLS, using the certificates obtained earlier.

Obtaining a certificate

When obtaining a certificate, there are two options: creating a self-signed certificate (a certificate signed by itself, not an actual CA), or creating a *certificate request* and having a reputable CA sign that request so it becomes a certificate.

The *crypto-utils* package contains a utility called **genkey** that supports both methods. To create a certificate (signing request) with **genkey**, run the following command, where ***<FQDN>*** is the fully qualified domain name clients will use to connect to your server:

```
[root@server1 ~]# genkey <FQDN>
```

During the creation, **genkey** will ask for the desired key size (choose at least **2048** bits), if a signing request should be made (answering no will create a self-signed certificate), whether the private key should be protected with a passphrase, and general information about the identity of the server.

After the process has completed, a number of files will be generated:

- **/etc/pki/tls/private/<*fqdn*>.key**: This is the private key. The private key should be kept at **0600** or **0400** permissions, and an SELinux context of **cert_t**. This key file should never be shared with the outside world.
- **/etc/pki/tls/certs/<*fqdn*>.0.csr**: This file is only generated if you requested a signing request. This is the file that you send to your CA to get it signed. You *never* need to send the private key to your CA.
- **/etc/pki/tls/certs/<*fqdn*>.crt**: This is the public certificate. This file is only generated when a self-signed certificate is requested. If a signing request was requested and sent to a CA, this is the file that will be returned from the CA. Permissions should be kept at **0644**, with an SELinux context of **cert_t**.

Install Apache HTTPD modules

Apache HTTPD needs an extension module to be installed to activate TLS support. On Red Hat Enterprise Linux 7, you can install this module using the *mod_ssl* package.

This package will automatically enable **httpd** for a default virtual host listening on port **443/TCP**. This default virtual host is configured in the file **/etc/httpd/conf.d/ssl.conf**.

Configure a virtual host with TLS

Virtual hosts with TLS are configured in the same way as regular virtual hosts, with some additional parameters. It is possible to use name-based virtual hosting with TLS, but some older browsers are not compatible with this approach.

The following is a simplified version of **/etc/httpd/conf.d/ssl.conf**:

```
Listen 443 https (1)

SSLPassPhraseDialog exec:/usr/libexec/httpd-ssl-pass-dialog (2)
SSLSessionCache         shmcb:/run/httpd/sslcache(512000)
SSLSessionCacheTimeout  300
SSLRandomSeed startup file:/dev/urandom  256
SSLRandomSeed connect builtin
SSLCryptoDevice builtin

<VirtualHost _default_:443> (3)
  ErrorLog logs/ssl_error_log
  TransferLog logs/ssl_access_log
  LogLevel warn

  SSLEngine on (4)

  SSLProtocol all -SSLv2 (5)

  SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5 (6)

  SSLCertificateFile /etc/pki/tls/certs/localhost.crt (7)

  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key (8)
  CustomLog logs/ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>
```

1. This directive instructs **`https`** to listen on port **`443/TCP`**. The second argument (**`https`**) is optional, since **`https`** is the default protocol for port **`443/TCP`**.
2. If the private key is encrypted with a passphrase, **`httpd`** needs a method of requesting a passphrase from a user at the console at startup. This directive specifies what program to execute to retrieve that passphrase.
3. This is the virtual host definition for a catch-all virtual host on port **`443/TCP`**.
4. This is the directive that actually turns on **`TLS`** for this virtual host.
5. This directive specifies the list of protocols that **`httpd`** is willing to speak with clients. For added security, the older, unsafe **`SSLv3`** protocol should also be disabled:

```
SSLProtocol all -SSLv2 -SSLv3
```

6. This directive lists what encryption ciphers **`httpd`** is willing to use when communicating with clients. The selection of ciphers can have big impacts on both performance and security.
7. This directive instructs **`httpd`** where it can read the certificate for this virtual host.
8. This directive instructs **`httpd`** where it can read the private key for this virtual host. **`httpd`** reads all private keys before privileges are dropped, so file permissions on the private key can remain locked down.

If a certificate signed by an CA is used, and the certificate itself does not have copies of all the CA certificates used in signing, up to a root CA, embedded in it, the server will also need to provide a *certificate chain*, a copy of all CA certificates used in the signing process concatenated together. The **`SSLCertificateChainFile`** directive is used to identify such a file.

When defining a new TLS-encrypted virtual host, it is not needed to copy the entire contents of **`ssl.conf`**. Only a **`<VirtualHost>`** block with the **`SSLEngine On`** directive, and configuration for certificates, is strictly needed. The following is an example of a name-based TLS virtual host:

```
<VirtualHost *:443>
  ServerName demo.example.com
  SSLEngine on
  SSLCertificateFile /etc/pki/tls/certs/demo.example.com.crt
  SSLCertificateKeyFile /etc/pki/tls/private/demo.example.com.key
  SSLCertificateChainFile /etc/pki/tls/certs/example-ca.crt
</VirtualHost>
```

This example misses some important directives such as **`DocumentRoot`**; these will be inherited from the main configuration.

Warning

Not defining what protocols and ciphers can be used will result in **`httpd`** using default options for these. **`httpd`** defaults are not considered secure, and it is highly recommended to restrict both to a more secure subset.

Configuring forward secrecy

If a weaker encryption cipher has been used, and the private key of the server has been compromised—for example, after a server break-in or a bug in the crypto code—an attacker could possibly decrypt a recorded session.

Protecting against these types of attacks is called ensuring *forward secrecy*. Forward secrecy can be established by carefully tuning the allowed ciphers in the **`SSLCipherSuite`** directive, and

having the server always select the most preferred cipher from this list that both the server and the client support.

The following is an example that, at the date of publication, was considered the best set of ciphers to allow. This list prioritizes ciphers that perform the initial session key exchange using elliptic curve Diffie-Hellman (EECDH) algorithms. Using Diffie-Hellman, the actual session key is never transmitted, but rather calculated by both sides.

```
SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384 EECDH+ECDSA
+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 EECDH EDH+aRSA RC4 !aNULL !
eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS"
SSLHonorCipherOrder On
```

The **SSLHonorCipherOrder On** directive instructs **httpd** to always prefer ciphers listed earlier on in the **SSLCipherSuite** list, regardless of the client preference.

Important

Security research is an always ongoing arms race. It is recommended that administrators re-evaluate their selected ciphers on a regular basis.

Configuring HTTP Strict Transport Security (HSTS)

A common misconfiguration, and one that will result in warnings in most modern browsers, is having a web page that is served out over **https** include resources served out over clear-text **http**.

To protect against this type of misconfiguration, add the following line inside a **<VirtualHost>** block that has TLS enabled:

```
Header always set Strict-Transport-Security "max-age=15768000"
```

Sending this extra header informs clients that they are not allowed to fetch any resources for this page that are not served using TLS.

Another possible issue comes from clients connecting over **http** to a resource they should have been using **https** for.

Simply not serving any content over **http** would alleviate this issue, but a more subtle approach is to automatically redirect clients connecting over **http** to the same resource using **https**.

To set up these redirects, configure a **http** virtual host for the same **ServerName** and **ServerAlias** as the TLS protected virtual host (a catch-all virtual host can be used), and add the following lines inside the **<VirtualHost *:80>** block:

```
RewriteEngine on
RewriteRule ^(/.*)$ https://%{HTTP_POST}$1 [redirect=301]
```

The **RewriteEngine on** directive turns on the URL rewrite module for this virtual host, and the **RewriteRule** matches any resource (**^(/.*)$**) and redirects it using a http **Moved Permanently** message (**[redirect=301]**) to the same resource served out over https. The **%{HTTP_HOST}** variable uses the hostname that was requested by the client, while the **$1** part

is a back-reference to whatever was matched between the first set of parentheses in the regular expression.

References

`httpd`(8) man page

httpd-manual contents

> Qualys SSL Labs: SSL/TLS Deployment Best Practices
> https://www.ssllabs.com/projects/best-practices/

Practice: Configuring a TLS-enabled Virtual Host

Guided exercise

In this lab, you will configure a TLS-encrypted virtual host.

Resources:	
Files:	• `http://classroom.example.com/pub/example-ca.crt` • `http://classroom.example.com/pub/tls/certs/www1.crt` • `http://classroom.example.com/pub/tls/private/www1.key`
Machines:	**`desktop1`** and **`server1`**

Outcomes:
A web server configured with two virtual hosts, **`www1.example.com`** protected by TLS.

Before you begin...
- Reset your **`server1`** system.

Your company has decided to start selling Jim Whitehurst action figures online. Since most Red Hat fans enjoy privacy and security, the website will need to be protected with TLS.

You have been asked to configure a web server on your **`server1`** machine to host this site. This web server will need to host the virtual host: `https://www1.example.com` The non-encrypted version of this site should send browsers an automatic redirect to the encrypted version.

Certificates and private keys for this site has already been provided. The certificates can be downloaded from `http://classroom.example.com/pub/tls/certs/www1.crt`, and the private keys can be found at `http://classroom.example.com/pub/tls/private/www1.key`. The public part of the signing CA can be found at `http://classroom.example.com/pub/example-ca.crt`.

Content for the site should be served out of **`/srv/www1/www`**. Since your web designers are currently on a two-week lunch break, you will have to provide temporary content that uniquely identifies each host yourself.

Custom logfiles are not required for now.

1. Install both the *httpd* and *mod_ssl* packages.
   -
```
[root@server1 ~]# yum install httpd mod_ssl
```
2. Create the content directories, with identifying content and appropriate SELinux contexts.
   2.1. Create the content directory.
```
[root@server1 ~]# mkdir -p /srv/www1/www
```

2.2. In the content directory, create an **index.html** file with distinct content.

```
[root@server1 ~]# echo "www1" > /srv/www1/www/index.html
```

2.3. Add a valid httpd content context for the new directory.

```
[root@server1 ~]# semanage fcontext -a -t httpd_sys_content_t '/srv(/.*)?'
```

2.4. Reset the SELinux context on your new directories.

```
[root@server1 ~]# restorecon -Rv /srv/
```

3. Download all the needed certificates and private keys to their correct locations with their correct permissions.

3.1. Download the CA certificate used to sign your certificates.

```
[root@server1 ~]# cd /etc/pki/tls/certs
[root@server1 certs]# wget http://classroom.example.com/pub/example-ca.crt
```

3.2. While still in the **certs** directory, download the certificates for your virtual host.

```
[root@server1 certs]# wget http://classroom.example.com/pub/tls/certs/www1.crt
```

3.3. Switch to the **private** directory and download the private key. Do not forget to set the permissions on the private key to **0600**.

```
[root@server1 certs]# cd /etc/pki/tls/private
[root@server1 private]# wget http://classroom.example.com/pub/tls/private/
www1.key
[root@server1 private]# chmod 0600 w*1.key
```

4. Configure the TLS name-based virtual host for your **www1.example.com** domain in a new file called **/etc/httpd/conf.d/www1.conf**. You can use the existing **/etc/httpd/conf.d/ssl.conf** as a template, but if you do, do not forget to strip out all the content outside of the **<VirtualHost>** block.

Do not forget to add an automatic redirect from the non-TLS-based http site to the TLS-encrypted https site.

4.1. Create **/etc/httpd/conf.d/www1.conf** with the following content:

```
<VirtualHost *:443>
  ServerName www1.example.com
  SSLEngine On
  SSLProtocol all -SSLv2 -SSLv3
  SSLCipherSuite HIGH:MEDIUM:!aNull:!MD5
  SSLHonorCipherOrder on
  SSLCertificateFile /etc/pki/tls/certs/www1.crt
  SSLCertificateKeyFile /etc/pki/tls/private/www1.key
```

```
  SSLCertificateChainFile /etc/pki/tls/certs/example-ca.crt
  DocumentRoot /srv/www1/www
</VirtualHost>
```

4.2. Add a **<Directory>** block for **/srv/www1/www** to **/etc/httpd/conf.d/www1.conf** like the following:

```
<Directory /srv/www1/www>
  Require all granted
</Directory>
```

4.3. To accomplish the automatic redirect from http to https, add the following block to **/etc/httpd/conf.d/www1.conf**:

```
<VirtualHost *:80>
  ServerName www1.example.com
  RewriteEngine on
  RewriteRule ^(/.*)$ https://%{HTTP_HOST}$1 [redirect=301]
</VirtualHost>
```

5. Start and enable the **httpd.service**, and open the relevant firewall ports.

5.1. Start and enable **httpd.service**.

```
[root@server1 ~]# systemctl start httpd.service
[root@server1 ~]# systemctl enable httpd.service
```

5.2. Open both the **http** and **https** ports on the firewall.

```
[root@server1 ~]# firewall-cmd --permanent --add-service=http --add-
service=https
[root@server1 ~]# firewall-cmd --reload
```

6. Test your new configuration from your **desktop1** system. You will have to import the `http://classroom.example.com/pub/example-ca.crt` into the list of trusted CA certificates for your browser as part of this process.

Perform all of the following steps on your **desktop1** system.

6.1. Download the **example-ca.crt** certificate to your home directory.

```
[student@desktop1 ~]$ wget http://classroom.example.com/pub/example-ca.crt
```

6.2. Test access from the command line:

```
[student@desktop1 ~]$ curl -Ls --cacert example-ca.crt http://www1.example.com
www1
```

6.3. Launch firefox and open the Edit > Preferences dialog. Navigate to the Advanced > Certificates tab.

6.4. Click View Certificates, then use the Import button. Navigate to the file you just downloaded and click Open. In the resulting dialog, check Trust this CA to identify websites and click OK.

Close all open dialogs.

6.5. Point your browser at `http://www1.example.com`. Both should redirect to the **`https`** counterpart automatically, without a certificate warning.

Note

When troubleshooting a web server using firefox, it can be useful to empty the cache from within the Preferences dialog. The Clear Now button can be found in the Advanced > Network tab. If the cache is not cleared in between server restarts, firefox might show old, outdated information.

7. *Bonus question:*

Without further configuration, a visit to `http://server1.example.com` will also result in a redirect to **`https`**. Why is this, and how could you prevent this from happening?

Answer:

This happens because there is an explicit catch-all virtual host defined for **`*:80`**, resulting in the first virtual host for **`*:80`** being used as a default virtual host. Since this is the virtual host for your **`webapp1`** domain, the redirect rule is included.

This can be solved by defining either a **`<VirtualHost _default_:80>`** block, or by defining a **`<VirtualHost *:80>`** block in a location where it will be parsed before any other virtual hosts; for example, before the includes in **`/etc/httpd/conf/httpd.conf`** or as a separate file in **`/etc/httpd/conf.d/00-default.conf`**.

Integrating Dynamic Web Content

Objectives

After completing this section, students should be able to configure Apache **httpd** to serve dynamic database-driven web content.

Dynamic content

Most modern websites do not consist of purely static content. Most content served out is actually generated dynamically, on demand. Integrating dynamic content with Apache HTTPD can be done in numerous ways. This section describes a number of the most common ways, but more ways exist.

Common Gateway Interface

One of the oldest forms of generating dynamic content is by using *Common Gateway Interface* (CGI). When a CGI resource is requested, **httpd** does not simply read the resource and serve it out; instead, it executes the resource as a process, and serves the **stdout** of that process. Although CGI resources are mostly written in scripting languages like Perl, it is also quite common for CGI resources to be compiled C programs, or Java executables.

Information from the request (including client information) is made available to the CGI program using environment variables.

Configuring httpd for CGI

To have **httpd** treat a location as CGI executables, the following syntax is used in the **httpd** configuration.

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

This instructs **httpd** to redirect any request for files under the **/cgi-bin/** URI to the **/var/www/cgi-bin/** directory, and treat the files in that directory as executable scripts.

A number of caveats exist when using CGI:

- CGI scripts will be executed as the **apache** user and group.
- CGI scripts should be executable by the **apache** user and group.
- CGI scripts should have the **httpd_sys_script_exec_t** SELinux context.
- The CGI directory should have **Options None**, and access should be granted using a normal **<Directory>** block.

Serving dynamic PHP content

A popular method of providing dynamic content is using the PHP scripting language. While PHP scripts can be served using old-fashioned CGI, both performance and security can be improved by having **httpd** run a PHP interpreter internally.

By installing the *php* package, a special **mod_php** module is added to **httpd**. The default configuration for this module adds the following lines to the main **httpd** configuration:

```
<FilesMatch \.php$>
  SetHandler application/x-httpd-php
<FilesMatch>
DirectoryIndex index.php
```

The **<FilesMatch>** block instructs **httpd** to use **mod_php** for any file with a name ending in **.php**, and the **DirectoryIndex** directive adds **index.php** to the list of files that will be sought when a directory is requested.

Serving dynamic Python content

Also popular is generating dynamic content using Python scripts. Python scripts can be served out using regular CGI, but both **python** and **httpd** also support a newer protocol: *Web Server Gateway Interface* (WSGI).

WSGI support can be added to **httpd** by installing the *mod_wsgi* package.

Unlike the **mod_php** or CGI approach, WSGI does not start a new script/interpreter for every request. Instead, a main application is started, and all requests are routed into that application.

Configuring **httpd** to support a WSGI application takes two steps:

1. Install the *mod_wsgi* package.
2. Add a **WSGIScriptAlias** line to a virtual host definition.

The following is an example of a **WSGIScriptAlias** directive, which sends all requests for `http://servername/myapp` and any resources below it to the WSGI application **/srv/myapp/www/myapp.py**:

```
WSGIScriptAlias /myapp/ /srv/myapp/www/myapp.py
```

WSGI applications should be executable by the **apache** user and group, and their SELinux contexts should be set to **httpd_sys_content_t**.

Database connectivity

Most web applications will need to store and retrieve persistent data. A common approach to this is to store the data in a database such as MariaDB or PostgreSQL.

When the database is running on the same host as the web server, and the database is using a standard network port, SELinux will allow the network connection from the web application to happen.

When a database on a remote host is used, the SELinux Boolean **httpd_can_network_connect_db** must be set to **1** to allow the connection.

When a network connection to another needs to be made from within the web application, and the target is not a well-known database port, the SELinux Boolean **httpd_can_network_connect** must be set to **1**.

Various other SELinux Booleans can also affect the way in which web applications are executed by **httpd**.

References

`httpd`(8) and **`httpd_php_selinux`**(8) man pages

httpd-manual package contents

`/usr/share/doc/mod_wsgi-*/README`

Summary

Configuring Apache HTTPD

In this section, students learned how to identify the key configuration files, log files, and content directories used by Apache **`httpd`**

Configuring and Troubleshooting Virtual Hosts

In this section, students learned how to configure Apache **`httpd`** to provide IP-based and name-based virtual hosts.

Configuring HTTPS

In this section, students learned how to configure Apache **`httpd`** to provide TLS-encrypted virtual hosts.

Integrating Dynamic Web Content

In this section, students learned how to configure Apache **`httpd`** to serve dynamic database-driven web content.

CHAPTER 10

WRITING BASH SCRIPTS

Overview	
Goal	To write simple, well-structured shell scripts using Bash's shell expansion features and for-loop construct.
Objectives	• To write simple shell scripts using Bash.
Sections	• Bash Shell Scripting Basics (and Practice)
Lab	• Writing Bash Scripts

Bash Shell Scripting Basics

Objectives

After completing this section, students should be able to write and debug simple Bash shell scripts which utilizes variable expansion, command line substitution, arithmetic expansion, and for loops..

Bash scripting basics

Many simple day-to-day system administration tasks can be accomplished by the numerous Linux command-line tools available to administrators. However, tasks with greater complexity often require the chaining together of multiple commands. In these situations, Linux command-line tools can be combined with the offerings of the Bash shell to create powerful shell scripts to solve real-world problems.

In its simplest form, a Bash shell script is simply an executable file composed of a list of commands. However, when well-written, a shell script can itself become a powerful command-line tool when executed on its own, and can even be further leveraged by other scripts.

Proficiency in shell scripting is essential to the success of Linux system administrators in all operational environments. Working knowledge of shell scripting is especially crucial in enterprise environments, where its use can translate to improved efficiency and accuracy of routine task completion.

Choosing a programming language

While Bash shell scripting can be used to accomplish many tasks, it may not be the right tool for all scenarios. Administrators have a wide variety of programming languages at their disposal, such as C, C++, Perl, Python, Ruby and other programming languages. Each programming language has its strengths and weaknesses, and as such, none of the programming languages are the right tool for every situation.

Bash shell scripts are a good choice for tasks which can be accomplished mainly by calling other command-line utilities. If the task involves heavy data processing and manipulation, other languages such as Perl or Python will be better suited for the job. While Bash supports arithmetic operations, they are limited to simple integer arithmetic. For more complex arithmetic operations, C or C++ should be considered. If a solution requires the use of arrays. Bash is probably not the best tool. Bash has supported one-dimensional arrays for some time, and the latest version even supports associative arrays. However, Perl or Python have much better array functionality, with the ability to accommodate multidimensional arrays.

As administrators become proficient at shell scripting, they will gain more knowledge regarding its capabilities and limitations. This experience combined with exposure to other programming languages over time, will provide administrators with a better understanding of the advantages and disadvantages of each, and which problems each one is best suited for.

Creating and executing Bash shell scripts

A Bash shell script can be created by opening a new empty file in a text editor. While any text editor can be used, advanced editors, such as *vim* or *emacs*, understand Bash shell syntax and can provide color-coded highlighting. This highlighting can be a tremendous help for spotting syntactical errors, such as unpaired quotes, unclosed brackets, and other common blunders.

The command interpreter

The first line of a Bash shell script begins with '**#!**', also commonly referred to as a *sharp-bang*, or the abbreviated version, *sha-bang*. This two-byte notation is technically referred to as the *magic pattern*. It indicates that the file is an executable shell script. The path name that follows is the *command interpreter*, the program that should be used to execute the script. Since Bash shell scripts are to be interpreted by the Bash shell, they begin with the following first line.

```
#!/bin/bash
...
```

Executing a Bash shell script

After a Bash shell script is written, its file permissions and ownership need to be modified so that it is executable. Execute permission is modified with the **chmod** command, possibly in conjunction with the **chown** command to change the file ownership of the script accordingly. Execute permission should only be granted to the users that the script is intended for.

Once a Bash shell script is executable, it can be invoked by entering its name on the command line. If only the base name of the script file is entered, Bash will search through the directories specified in the shell's **PATH** environmental variable, looking for the first instance of an executable file matching that name. Administrators should avoid script names that match other executable files, and should also ensure that the **PATH** variable is correctly configured on their system so that the script will be the first match found by the shell. The **which** command, followed by the file-name of the executeable script displays in which directory the script resides that is executed when the script name is invoked as a command resides.

```
[student@server1 ~]$ which hello
~/bin/hello

[student@server1 ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/student/.local/bin:/home/
student/bin
```

Displaying output

The **echo** command can be used to display arbitrary text by passing the text as an argument to the command. By default, the text is directed to *standard out (STDOUT)*, but can also be directed to *standard error (STDERR)* using output redirection. In the following simple Bash script, the **echo** command displays the message "*Hello, world*" to *STDOUT*.

```
[student@server1 ~]$ cat hello
#!/bin/bash

echo "Hello, world"

[student@server1 ~]$ ./hello
Hello, world
```

While seemingly simple in its function, the **echo** command is widely used in shell scripts due to its usefulness for various purposes. It is commonly used to display informational or error messages during the script execution. These messages can be a helpful indicator of the progress of a script and can be directed either to standard out, standard error, or be redirected to a log file for archiving. When displaying error messages, it is good practice to direct them to *STDERR* to make it easier to differentiate error messages from normal status messages.

```
[student@server1 ~]$ cat hello
#!/bin/bash

echo "Hello, world"
echo "ERROR: Houston, we have a problem." >&2

[student@server1 ~]$ ./hello 2> hello.log
Hello, world

[student@server1 ~]$ cat hello.log
ERROR: Houston, we have a problem.
```

The **echo** command can also be very helpful when trying to debug a problematic shell script. The addition of **echo** statements to the portion of the script that is not behaving as expected can help clarify the commands being executed, as well as the values of variables being invoked.

Quoting special characters

A number of characters or words have special meanings to the Bash shell in specific context. There are situations when the literal values, rather than the special meanings, of these characters or words are desired. For example, the **#** character is interpreted by Bash as the beginning of a comment and is therefore ignored, along with everything following it on the same line. If this special meaning is not desired, then Bash needs to be informed that the **#** character is to be treated as a literal value. The meanings of special characters or words can be disabled through the use of the escape character, ****, single quotes, **''**, or double quotes, **""**.

The escape character, ****, removes the special meaning of the single character immediately following it. For example, to display the literal string *# test* with the **echo** command, the **#** character must not be interpreted by Bash with special meaning. The escape character can be placed in front of the **#** character to disable its special meaning.

```
[student@server1 ~]$ echo # not a comment

[student@server1 ~]$ echo \# not a comment
# not a comment
```

The escape character, ****, only removes the special meaning of a single character. When more than one character in a text string needs to be escaped, users can either use the escape character multiple times or employ single quotes, **''**. Single quotes preserve the literal meaning of all characters they enclose. The following example demonstrates how single quotes can be used when multiple characters need to be escaped.

```
[student@server1 ~]$ echo # not a comment #

[student@server1 ~]$ echo \# not a comment #
# not a comment

[student@server1 ~]$ echo \# not a comment \#
# not a comment #

[student@server1 ~]$ echo '# not a comment #'
# not a comment #
```

While single quotes preserve the literal value of all characters they enclose, double quotes differ in that they do not preserve the literal value of the dollar sign, **$**, the back-ticks, **``**, and the backslash, ****. When enclosed with double quotes, the dollar sign and back-ticks preserve their

special meaning, and the special meaning of the backslash character is only retained when it precedes a dollar sign, back-tick, double quote, backslash, or newline.

```
[student@server1 ~]$ echo '$HOME'
$HOME

[student@server1 ~]$ echo '`pwd`'
`pwd`

[student@server1 ~]$ echo '"Hello, world"'
"Hello, world"

[student@server1 ~]$ echo "$HOME"
/home/student

[student@server1 ~]$ echo "`pwd`"
/home/student

[student@server1 ~]$ echo ""Hello, world""
Hello, world

[student@server1 ~]$ echo "\$HOME"
$HOME

[student@server1 ~]$ echo "\`pwd\`"
`pwd`

[student@server1 ~]$ echo "\"Hello, world\""
"Hello, world"
```

Using variables

As the complexity of a shell script increases, it is often helpful to make use of variables. A variable serves as a container, within which a shell script can store data in memory. Variables make it easy to access and modify the stored data during a script's execution.

Assigning values to variables

Data is assigned as a value to a variable via the following syntax:

```
VARIABLENAME=value
```

While variable names are typically uppercase letters, they can be made up of numbers, letters (uppercase and lowercase), and the underscore character, '_'. However, a variable name cannot start with a number. The equal sign, **=**, is used to assign values to variables and must not be separated from the variable name or the value by spaces. The following are some examples of valid variable names.

```
COUNT=40
first_name=John
file1=/tmp/abc
_ID=RH123
```

Two common types of data stored in variables are integer values and string values. It is good practice to quote string values when assigning them to variables, since the space character is interpreted by Bash as a word separator when not enclosed within single or double quotes. Whether single or double quotes should be used to enclose variable values depends on how characters with special meanings to Bash should be treated.

```
full_name='John Doe'
full_name="$FIRST $LAST"
price='$1'
```

Expanding variable values

The value of a variable can be recalled through a process known as *variable expansion* by preceding the variable name with a dollar sign, **$**. For example, the value of the **VARIABLENAME** variable can be referenced with **$VARIABLENAME**. The **$VARIABLENAME** syntax is the simplified version of the brace-quoted form of variable expansion, **${VARIABLENAME}**. While the simplified form is usually acceptable, there are situations where the brace-quoted form must be used to remove ambiguity and avoid unexpected results.

In the following example, without the use of brace quotes, Bash will interpret **$FIRST_$LAST** as the variable **$FIRST**_ followed by the variable **$LAST**, rather than the variables **$FIRST** and **$LAST** separated by the '_' character. Therefore, brace quoting must be used for variable expansion to function properly in this scenario.

```
[student@server1 ~]$ FIRST_=Jane

[student@server1 ~]$ FIRST=John

[student@server1 ~]$ LAST=Doe

[student@server1 ~]$ echo $FIRST_$LAST
JaneDoe

[student@server1 ~]$ echo ${FIRST}_$LAST
John_Doe
```

Using Bash shell expansion features

Aside from variable expansion, the Bash shell offers several other types of shell expansion features. Of these, command substitution and arithmetic expansion can be useful in Bash shell scripting, and are commonly used.

Command substitution

Command substitution replaces the invocation of a command with the output from its execution. This feature allows the output of a command to be used in a new context, such as the argument to another command, the value for a variable, and the list for a loop construct.

Command substitution can be invoked with the old form of enclosing the command in back-ticks, such as **`<COMMAND>`**. However, the preferred method is to use the newer **$()** syntax, **$(<COMMAND>)**.

```
[student@server1 ~]$ echo "Current time: `date`"
Current time is Thu Jun  5 16:24:24 EDT 2014.

[student@server1 ~]$ echo "Current time: $(date)"
Current time is Thu Jun  5 16:24:30 EDT 2014.
```

The newer syntax is preferred since it allows for nesting of command substitutions. In the following nested command substitution example, the output of the **find** command is used as arguments for the tar command, which then has its output stored into the variable **TAROUTPUT**.

```
[root@server1 ~]# TAROUTPUT=$(tar cvf /tmp/incremental_backup.tar $(find /etc -type f -
mtime -1))

[root@server1 ~]# echo $TAROUTPUT
/etc/group /etc/gshadow /etc/shadow- /etc/passwd /etc/shadow /etc/passwd- /etc/tuned/
active_profile /etc/rht /etc/group- /etc/gshadow- /etc/resolv.conf
```

Arithmetic expansion

Bash's arithmetic expansion can be used to perform simple integer arithmetic operations, and uses the syntax **$[<EXPRESSION>]**. When enclosed within **$[]**, arithmetic expressions are evaluated by Bash and then replaced with their results. Bash performs variable expansion and command substitution on the enclosed expression before its evaluation. Like command line substitution, nesting of arithmetic substitutions is allowed.

```
[student@server1 ~]$ echo $[1+1]
2

[student@server1 ~]$ echo $[2*2]
4

[student@server1 ~]$ COUNT=1; echo $[$[$COUNT+1]*2]
4
```

Space characters are allowed in the expression used within an arithmetic expansion. The use of space characters can improve readability in complicated expressions or when variables are included.

```
[student@server1 ~]$ SEC_PER_MIN=60

[student@server1 ~]$ MIN_PER_HR=60

[student@server1 ~]$ HR_PER_DAY=24

[student@server1 ~]$ SEC_PER_DAY=$[ $SEC_PER_MIN * $MIN_PER_HR * $HR_PER_DAY ]

[student@server1 ~]$ echo "There are $SEC_PER_DAY seconds in a day."
There are 86400 seconds in a day.
```

The following are some of the commonly used operators in arithmetic expressions, along with their meanings.

Operator	Meaning
<VARIABLE>++	variable post-increment
<VARIABLE>--	variable post-decrement
++<VARIABLE>	variable pre-increment
--<VARIABLE>	variable pre-decrement
-	unary minus
+	unary plus
**	exponentiation
*	multiplication
/	division

Operator	Meaning
%	remainder
+	addition
-	subtraction

When multiple operators exist in an expression, Bash will evaluate certain operators in order according to their precedence. For example, multiplication and division operators have a higher precedence than addition and subtraction. Parentheses can be used to group sub expressions if the evaluation order desired differs from the default precedence.

```
[student@server1 ~]$ echo $[ 1 + 1 * 2]
3

[student@server1 ~]$ echo $[ (1 +1) * 2 ]
4
```

The following table lists the order of precedence for commonly used arithmetic operators from highest to lowest. Operators that have equal precedence are listed together.

Operator	Meaning
<VARIABLE>++, <VARIABLE>--	variable post-increment and post-decrement
++<VARIABLE>, --<VARIABLE>	variable pre-increment and pre-decrement
-, +	unary minus and plus
**	exponentiation
*, /, %	multiplication, division, remainder
+, -	addition, subtraction

Iterating with the for loop

System administrators often encounter repetitive tasks in their day-to-day activities. Repetitive tasks can take the form of executing an action multiple times on an target, such as checking a process every minute for 10 minutes to see if it has completed. Task repetition can also take the form of executing an action a single time across multiple targets, such as performing a database backup of each database on a system. The *for loop* is one of the multiple shell looping constructs offered by Bash, and can be used for task iterations.

Using the for loop

Bash's for-loop construct uses the following syntax. The loop processes the items provided in **<LIST>** in order one by one and exits after processing the last item on the list. Each item in the list is temporarily stored as the value of **<VARIABLE>**, while the for loop executes the block of commands contained in its construct. The naming of the variable is arbitrary. Typically, the variable value is referenced by commands in the command block.

```
for <VARIABLE> in <LIST>; do
     <COMMAND>
     ...
     <COMMAND> referencing <VARIABLE>
done
```

The list of items provided to a for loop can be supplied in several ways. It can be a list of items entered directly by the user, or be generated from different types of shell expansion, such as variable expansion, brace expansion, file name expansion, and command substitution. Some examples that demonstrate the different ways lists can be provided to for loops follow.

```
[student@server1 ~]$ for HOST in host1 host2 host3; do echo $HOST; done
host1
host2
host3

[student@server1 ~]$ for HOST in host{1,2,3}; do echo $HOST; done
host1
host2
host3

[student@server1 ~]$ for HOST in host{1..3}; do echo $HOST; done
host1
host2
host3

[student@server1 ~]$ for FILE in file*; do ls $FILE; done
filea
fileb
filec

[student@server1 ~]$ for FILE in file{a..c}; do ls $FILE; done
filea
fileb
filec

[student@server1 ~]$ for PACKAGE in $(rpm -qa | grep kernel); do echo "$PACKAGE was
 installed on $(date -d @$(rpm -q --qf "%{INSTALLTIME}\n" $PACKAGE))"; done
abrt-addon-kerneloops-2.1.11-12.el7.x86_64 was installed on Tue Apr 22 00:09:07 EDT 2014
kernel-3.10.0-121.el7.x86_64 was installed on Thu Apr 10 15:27:52 EDT 2014
kernel-tools-3.10.0-121.el7.x86_64 was installed on Thu Apr 10 15:28:01 EDT 2014
kernel-tools-libs-3.10.0-121.el7.x86_64 was installed on Thu Apr 10 15:26:22 EDT 2014

[student@server1 ~]$ for EVEN in $(seq 2 2 8); do echo "$EVEN"; done; echo "Who do we
 appreciate?"
2
4
6
8
Who do we appreciate?
```

Troubleshooting shell script bugs

Inevitably, administrators who write, use, or maintain shell scripts will encounter bugs with a script. Bugs are typically due to typographical errors, syntactical errors, or poor script logic.

A good way to deal with shell scripting bugs is to make a concerted effort to prevent them from occurring in the first place during the authoring of the script. As previously mentioned, using a text editor with Bash syntactical highlighting can help make mistakes more obvious when writing scripts. Another easy way to avoid introducing bugs into scripts is by adhering to good practices during the creation of the script.

Good styling practices

Use comments to help clarify to readers the purpose and logic of the script. The top of every script should include comments providing an overview of the script's purpose, intended actions, and

general logic. Also use comments throughout the script to clarify the key portions, and especially sections that may cause confusion. Comments will not only aid other users in the reading and debugging of the script, but will also often help the author recall the workings of the script once some time has passed.

Structure the contents of the script to improve readability. As long as the syntax is correct, the command interpreter will flawlessly execute the commands within a script with absolutely no regard for their structure or formatting. Here are some good practices to follow:

- Break up long commands into multiple lines of smaller code chunks. Shorter pieces of code are much easier for readers to digest and comprehend.
- Line up the beginning and ending of multiline statements to make it easier to see that control structures begin and end, and whether they are being closed properly.
- Indent lines with multiline statements to represent the hierarchy of code logic and the flow of control structures.
- Use line spacing to separate command blocks to clarify when one code section ends and another begins.
- Use consistent formatting through the entirety of a script.

When utilized, these simple practices can make it significantly easier to spot mistakes during authoring, as well as improve the readability of the script for future readers. The following example demonstrates how the incorporation of comments and spacing can greatly improve script readability.

```
#!/bin/bash
for PACKAGE in $(rpm -qa | grep kernel); do echo "$PACKAGE was installed on $(date -d @
$(rpm -q --qf "%{INSTALLTIME}\n" $PACKAGE))"; done
```

```
#!/bin/bash
#
# This script provides information regarding when kernel-related packages
# are installed on a system by querying information from the RPM database.
#

# Variables
PACKAGETYPE=kernel
PACKAGES=$(rpm -qa | grep $PACKAGETYPE)

# Loop through packages
for PACKAGE in $PACKAGES; do
     # Determine package install date and time
     INSTALLEPOCH=$(rpm -q --qf "%{INSTALLTIME}\n" $PACKAGE)

     # RPM reports time in epoch, so need to convert
     # it to date and time format with date command
     INSTALLDATETIME=$(date -d @$INSTALLEPOCH)

     # Print message
     echo "$PACKAGE was installed on $INSTALLDATETIME"
done
```

Do not make assumptions regarding the outcome of actions taken by a script. This is especially true of inputs to the script, such as command-line arguments, input from users, command

substitutions, variable expansions, and file name expansions. Rather than making assumptions about the integrity of these inputs, make the worthwhile effort to employ the use of proper quoting and sanity checking.

The same caution should be utilized when acting upon entities external to the script. This includes interacting with files, and calling external commands. Make use of Bash's vast number of file and directory tests when interacting with files and directories. Perform error checking on the exit status of commands rather than counting on their success and blindly continuing along with the script when an unexpected error occurs.

The extra steps taken to rule out assumptions will increase the script's robustness, and keep it from being easily derailed and then inflicting unintended and unnecessary damage to a system. A couple of seemingly harmless lines of code, such as the ones that follow, make very risky assumptions about command execution outcome and file name expansion. If the directory change fails, either due to directory permissions or the directory being nonexistent, the subsequent file removal will be performed on a list of unknown files in an unintended directory.

```
cd $TMPDIR
rm *
```

Lastly, while well-intentioned administrators may employ good practices when authoring their scripts, not all will always agree on what constitutes good practices. Administrators should do themselves and others a favor and always apply their practices consistently through the entirety of their scripts. They should also be considerate and understanding of individual differences when it comes to programming styles and formatting in scripts authored by others. When modifying others' scripts, administrators should follow the existing structure, formatting, and practices used by the original author, rather than imposing their own style on a portion of the script and destroying the script's consistency, and thereby ruining its readability and future maintainability.

Debug and verbose modes

If despite best efforts, bugs are introduced into a script, administrators will find Bash's debug mode extremely useful. To activate the debug mode on a script, add the **-x** option to the command interpreter in the first line of the script.

```
#!/bin/bash -x
```

Another way to run a script in debug mode is to execute the script as an argument to Bash with the **-x** option.

```
[student@server1 bin]$ bash -x <SCRIPTNAME>
```

Bash's debug mode will print out commands executed by the script prior to their execution. The results of all shell expansion performed will be displayed in the printout. The following example shows the extra output that is displayed when debug mode is activated.

```
[student@server1 bin]$ cat filesize
#!/bin/bash

DIR=/home/student/tmp

for FILE in $DIR/*; do
 echo "File $FILE is $(stat --printf='%s' $FILE) bytes."
done
```

```
[student@server1 bin]$ ./filesize
File /home/student/tmp/filea is 133 bytes.
File /home/student/tmp/fileb is 266 bytes.
File /home/student/tmp/filec is 399 bytes.

[student@server1 bin]$ bash -x ./filesize
+ DIR=/home/student/tmp
+ for FILE in '$DIR/*'
++ stat --printf=%s /home/student/tmp/filea
+ echo 'File /home/student/tmp/filea is 133 bytes.'
File /home/student/tmp/filea is 133 bytes.
+ for FILE in '$DIR/*'
++ stat --printf=%s /home/student/tmp/fileb
+ echo 'File /home/student/tmp/fileb is 266 bytes.'
File /home/student/tmp/fileb is 266 bytes.
+ for FILE in '$DIR/*'
++ stat --printf=%s /home/student/tmp/filec
+ echo 'File /home/student/tmp/filec is 399 bytes.'
File /home/student/tmp/filec is 399 bytes.
```

While Bash's debug mode provides helpful information, the voluminous output may actually become more hindrance than help for troubleshooting, especially as the lengths of scripts increase. Fortunately, the debug mode can be enabled partially on just a portion of a script, rather than on its entirety. This feature is especially useful when debugging a long script and the source of the problem has been narrowed to a portion of the script.

Debugging can be turned on at a specific point in a script by inserting the command **`set -x`** and turned off by inserting the command **`set +x`**. The following demonstration shows the previous example script with debugging enabled just for the command line enclosed in the for loop.

```
[student@server1 bin]$ cat filesize
#!/bin/bash

DIR=/home/student/tmp

for FILE in $DIR/*; do
        set -x
        echo "File $FILE is $(stat --printf='%s' $FILE) bytes."
        set +x
done

[student@server1 bin]$ ./filesize
++ stat --printf=%s /home/student/tmp/filea
+ echo 'File /home/student/tmp/filea is 133 bytes.'
File /home/student/tmp/filea is 133 bytes.
+ set +x
++ stat --printf=%s /home/student/tmp/fileb
+ echo 'File /home/student/tmp/fileb is 266 bytes.'
File /home/student/tmp/fileb is 266 bytes.
+ set +x
++ stat --printf=%s /home/student/tmp/filec
+ echo 'File /home/student/tmp/filec is 399 bytes.'
File /home/student/tmp/filec is 399 bytes.
+ set +x
```

In addition to debug mode, Bash also offers a verbose mode, which can be invoked with the **`-v`** option. In verbose mode, Bash will print each command to standard out prior to its execution.

```
[student@server1 bin]$ cat filesize
```

```
#!/bin/bash

DIR=/home/student/tmp

for FILE in $DIR/*; do
 echo "File $FILE is $(stat --printf='%s' $FILE) bytes."
done

[student@server1 bin]$ bash -v ./filesize
stat --printf='%s' $FILE) bytes."
stat --printf='%s' $FILE) bytes.
stat --printf='%s' $FILE
File /home/student/tmp/filea is 133 bytes.
stat --printf='%s' $FILE) bytes."
stat --printf='%s' $FILE) bytes.
stat --printf='%s' $FILE
File /home/student/tmp/fileb is 266 bytes.
stat --printf='%s' $FILE) bytes."
stat --printf='%s' $FILE) bytes.
stat --printf='%s' $FILE
File /home/student/tmp/filec is 399 bytes.
```

Like the debug feature, the verbose feature can also be turned on and off at specific points in a script by inserting the **set -v** and **set +v** lines, respectively.

References

bash(1), **magic**(5), **echo**(1), **echo**(1p), and **seq**(1) man pages

Practice: Writing Bash Scripts

Guided exercise

In this lab, you will create a Bash shell script to automate the process of individually backing up every MariaDB database on **server1** and generating a report providing statistics on each database backup.

Resources:	
Machines:	**server1**

Outcomes:
A Bash script which automates the process of performing individual backups of each database in MariaDB. The script will also report the size statistics on each database backup.

Before you begin...
- Reset **server1**.
- Log into **server1** and become root with **su**.

You have been tasked with writing a Bash shell script to perform a backup of all MariaDB databases on a server using **mysqldump**. Each database will be backed up to the directory **/dbbackup** and will be named **DATABASENAME.dump**. Your script will print out the message **'Backing up "DATABASENAME"'** as it initiates the dump of each database.

Once the backups for all the databases are completed, your script will generate a report showing the name of each database backup, its size, and the percentage of the total database dump size it accounts for. The data for each database should be lined up in columns for readability.

The complete output of your script should look like the following:

```
[root@server1 ~]# /usr/local/sbin/dbbackup
Backing up "mysql"
Backing up "test"

/dbbackup/mysql.dump             514664 99%
/dbbackup/test.dump                1261  0%
```

1. Install the *mariadb-server* package, then enable and start the **mariadb** service.

   1.1. Install *mariadb-server* package with **yum**.

   ```
   [root@server1 ~]# yum install -y mariadb-server
   ```

   1.2. Enable and start **mariadb**.

   ```
   [root@server1 ~]# systemctl enable mariadb
   ln -s '/usr/lib/systemd/system/mariadb.service' '/etc/systemd/system/multi-
   user.target.wants/mariadb.service'
   [root@server1 ~]# systemctl start mariadb
   ```

2. Create the database backup directory.

```
[root@server1 ~]# mkdir /dbbackup
```

3. On the command line, formulate a command to generate a list of database names, excluding the system databases **information_schema** and **performance_schema**.

   3.1. Issue the **SHOW DATABASES** command to **mysql** as the **root** MySQL user.

```
[root@server1 ~]# mysql -u root -e 'SHOW DATABASES'
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

   3.2. Utilize the **--skip-column-names** and **-E** formatting options to simplify the output for parsing.

```
[root@server1 ~]# mysql --skip-column-names -E -u root -e 'SHOW DATABASES'
*************************** 1. row ***************************
information_schema
*************************** 2. row ***************************
mysql
*************************** 3. row ***************************
performance_schema
*************************** 4. row ***************************
test
```

   3.3. Exclude the row header lines and the two system databases from the output.

```
[root@server1 ~]# mysql --skip-column-names -E -u root -e 'SHOW DATABASES' |
 grep -v '^*' | grep -v '^information_schema$' | grep -v '^performance_schema$'
mysql
test
```

4. Create your script. Store the MySQL user, the formatting options, the **SHOW DATABASES** command, and the backup directory as variables.

   4.1. Create the new script file with a text editor.

```
[root@server1 ~]# vim /usr/local/sbin/dbbackup
```

   4.2. Specify the interpreter program for the script.

```
#!/bin/bash
```

   4.3. Set the variables.

```
# Variables
DBUSER=root
```

```
FMTOPTIONS='--skip-column-names -E'
COMMAND='SHOW DATABASES'
BACKUPDIR=/dbbackup
```

5. Initiate a for loop and loop through the list of databases to back up each one to the **/dbbackup** directory.

   5.1. Initiate the for loop by passing in a list of database names via command substitution.

```
# Backup non-system databases
for DBNAME in $(mysql $FMTOPTIONS -u $DBUSER -e "$COMMAND" | grep -v ^* | grep -
v information_schema | grep -v performance_schema); do
```

   5.2. Add the commands to be executed within each loop.

```
  echo "Backing up \"$DBNAME\""
  mysqldump -u $DBUSER $DBNAME > $BACKUPDIR/$DBNAME.dump
```

   5.3. Close the for loop.

```
done
```

6. Generate a report of each database's name, dump size, and the percentage of the total dump size it accounts for.

   6.1. Initiate a for loop to iterate through and total up the size of each database dump in the **/dbbackup** directory.

```
# Add up size of all database dumps
for DBDUMP in $BACKUPDIR/*; do
```

   6.2. Add the commands to be executed within each loop.

```
  SIZE=$(stat --printf "%s\n" $DBDUMP)
  TOTAL=$[ $TOTAL + $SIZE ]
```

   6.3. Close the for loop.

```
done
```

   6.4. Create a for loop to iterate through and report on each database dump.

```
# Report name, size, and percentage of total for each database dump
echo
for DBDUMP in $BACKUPDIR/*; do
```

   6.5. Add the commands to be executed within each loop.

```
  SIZE=$(stat --printf "%s\n" $DBDUMP)
  echo "$DBDUMP,$SIZE,$[ 100 * $SIZE / $TOTAL ]%"
```

6.6. Close the for loop.

```
done
```

7. Save and execute the script.

7.1. Make the script executable.

```
[root@server1 ~]# chmod u+x /usr/local/sbin/dbbackup
```

7.2. Execute the script.

```
[root@server1 ~]# /usr/local/sbin/dbbackup
Backing up "mysql"
Backing up "test"

/dbbackup/mysql.dump,514664,99%
/dbbackup/test.dump,1261,0%
```

Summary

Bash Shell Scripting Basics

In this section, students learned how to:

- Write, execute, and debug simple Bash shell scripts.
- Store and retrieve data during script execution with the use of variables.
- Enhance script functionality with command substitution and arithmetic expansion.
- Perform iterative execution of tasks with for loops.
- Troubleshoot Bash shell script issues with debug and verbose modes.

CHAPTER 11

BASH CONDITIONALS AND CONTROL STRUCTURES

Overview	
Goal	To use Bash conditionals and other control structures to write more sophisticated shell commands and scripts.
Objectives	• Incorporate the use of positional parameters, exit status, test conditions, and conditional structures to implement flow control in Bash shell scripts.
Sections	• Enhancing Bash Shell Scripts with Conditionals and Control Structures (and Practice)
Lab	• Bash Conditionals and Control Structures

Enhancing Bash Shell Scripts with Conditionals and Control Structures

Objectives

After completing this section, students should be able to incorporate the use of positional parameters, exit status, test conditions, and conditional structures to implement flow control in Bash shell scripts.

Using Bash special variables

While user-defined variables provide a means for script authors to create containers to store values used by a script, Bash also provides some predefined variables, which can be useful when writing shell scripts. One type of predefined variable is positional parameters.

Positional parameters

Positional parameters are variables which store the values of command-line arguments to a script. The variables are named numerically. The variable **0** refers to the script name itself. Following that, the variable **1** is predefined with the first argument to the script as its value, the variable **2** contains the second argument, and so on. The values can be referenced with the syntax **$1**, **$2**, etc.

Important

While rare, when referencing values past the ninth positional parameter, the brace-quoted form of variable expansion must be used. For example, the value of the tenth positional argument must be referenced with the syntax **${10}** rather than **$10**. Otherwise, Bash will expand the '**$1**' in **$10** to the value of the first positional argument to the script.

Bash provides special variables to refer to positional parameters: **$*** and **$@**. Both of these variables refer to all arguments in a script, but with a slight difference. When **$*** is used, all of the arguments are seen as a single word. However, when **$@** is used, each argument is seen as a separate word. This is demonstrated in the following example.

```
[student@server1 bin]$ cat showargs
#!/bin/bash

for ARG in "$*"; do
 echo $ARG
done

[student@server1 bin]$ ./showargs 1 2 3
1 2 3

[student@server1 bin]$ cat showargs
#!/bin/bash

for ARG in "$@"; do
 echo $ARG
done

[student@server1 bin]$ ./showargs 1 2 3
```

```
1
2
3
```

Another value which may be useful when working with positional parameters is **$#**, which represents the number of command-line arguments passed to a script. This value can be used to verify whether any arguments, or the correct number of arguments, are passed to a script.

```
[student@server1 bin]$ cat countargs
#!/bin/bash
echo "There are $# arguments."

[student@server1 bin]$ ./countargs
There are 0 arguments.

[student@server1 bin]$ ./countargs 1 2 3
There are 3 arguments.
```

Evaluating exit codes

Every command returns an exit status, also commonly referred to as return status or exit code. A successful command exits with an exit status of **0**. Unsuccessful commands exit with a nonzero exit status. Upon completion, a command's exit status is passed to the parent process and stored in the **?** variable. Therefore, the exit status of an executed command can be retrieved by displaying the value of **$?**. The following examples demonstrate the execution and exit status retrieval of several common commands.

```
[student@server1 bin]$  ls /etc/hosts
/etc/hosts

[student@server1 bin]$ echo $?
0

[student@server1 bin]$ ls /etc/nofile
ls: cannot access /etc/nofile: No such file or directory

[student@server1 bin]$ echo $?
2

[student@server1 bin]$ grep localhost /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

[student@server1 bin]$ echo $?
0

[student@server1 bin]$ grep random /etc/hosts

[student@server1 bin]$ echo $?
1
```

Using exit codes within a script

Once executed, a script will exit when it has processed all of its contents. However, there may be times when it is desirable to exit a script midway through, such as when an error condition is encountered. This can be accomplished with the use of the **exit** command within a script. When a script encounters the **exit** command, it will exit immediately and skip the processing of the remainder of the script.

The **exit** command can be executed with an optional integer argument between **0** and **255**, which represents an exit code. An exit code value of **0** represents no error. All other nonzero values indicate an error exit code. Script authors can use different nonzero values to differentiate between different types of errors encountered. This exit code is passed back to the parent process, which stores it in the **?** variable and can be accessed with **$?** as demonstrated in the following examples.

```
[student@server1 bin]$ cat hello
#!/bin/bash
echo "Hello, world"
exit 0

[student@server1 bin]$ ./hello
Hello, world

[student@server1 bin]$ echo $?
0
```

```
[student@server1 bin]$ cat hello
#!/bin/bash
echo "Hello, world"
exit 1

[student@server1 bin]$ ./hello
Hello, world

[student@server1 bin]$ echo $?
1
```

If the **exit** command is called without an argument, then the script will exit and pass on to the parent process the exit status of the last command executed.

Testing script inputs

To ensure that scripts are not easily derailed by unexpected conditions, it is good practice for script authors to not make assumptions regarding inputs, such as command-line arguments, user inputs, command substitutions, variable expansions, file name expansions, etc. Integrity checking can be performed by using Bash's test feature. Tests can be performed using Bash's test command syntax, **[<TESTEXPRESSION>]**. They can also be performed using Bash's newer extended test command syntax, **[[<TESTEXPRESSION>]]**, which has been available since Bash version 2.02.

Like all commands, the **test** command produces an exit code upon completion, which is stored as the value **$?**. To see the conclusion of a test, simply display the value of **$?** immediately following the execution of the **test** command. Once again, an exit status value of **0** indicates the test succeeded, while nonzero values indicate the test failed.

Performing comparison tests

Comparison test expressions make use of *binary comparison operators*. These operators expect two objects, one on each side of the operator, and evaluate the two for equality and inequality. Bash uses a different set of operators for string and numeric comparisons, and uses the following syntax format:

```
[ <ITEM1> <BINARY COMPARISON OPERATOR> <ITEM2> ]
```

Bash's numeric comparison is limited to integer comparison. The following list of binary comparison operators is used in Bash for integer comparison.

Operator	Meaning	Example
-eq	is equal to	["$a" -eq "$b"]
-ne	is not equal to	["$a" -ne "$b"]
-gt	is greater than	["$a" -gt "$b"]
-ge	is greater than or equal to	["$a" -ge "$b"]
-lt	is less than	["$a" -lt "$b"]
-le	is less than or equal to	["$a" -le "$b"]

The following examples demonstrate the use of Bash's numeric comparison operators.

```
[student@server1 ~]$ [ 1 -eq 1 ]; echo $?
0
[student@server1 ~]$ [ 1 -ne 1 ]; echo $?
1
[student@server1 ~]$ [ 8 -gt 2 ]; echo $?
0
[student@server1 ~]$ [ 2 -ge 2 ]; echo $?
0
[student@server1 ~]$ [ 2 -lt 2 ]; echo $?
1
[student@server1 ~]$ [ 1 -lt 2 ]; echo $?
0
```

Bash's string comparison uses the following binary operators.

Operator	Meaning	Example
=	is equal to	["$a" = "$b"]
==	is equal to	["$a" == "$b"]
!=	is not equal to	["$a" != "$b"]

The following examples demonstrate the use of Bash's string comparison operators.

```
[student@server1 ~]$ [ abc = abc ]; echo $?
0
[student@server1 ~]$ [ abc == def ]; echo $?
1
[student@server1 ~]$ [ abc != def ]; echo $?
0
```

Bash also has a few unary operators available for string evaluation. Unary operators evaluate just one item using the following format.

```
[ <UNARY OPERATOR> <ITEM> ]
```

The following table shows Bash's unary operators for string evaluation.

Operator	Meaning	Example
-z	string is zero length (null)	[-z "$a"]
-n	string is not null	[-n "$a"]

The following examples demonstrate the use of Bash's string unary operators.

```
[student@server1 ~]$ STRING=''; [ -z "$STRING" ]; echo $?
0
[student@server1 ~]$ STRING='abc'; [ -n "$STRING" ]; echo $?
0
```

Testing files and directories

Bash's string and binary operators allow users to implement the good practice of not assuming the integrity of inputs to a shell script. The same caution should be utilized when scripts interact with external entities, such as files and directories. Bash offers a large number of test operators for this purpose, as listed in the following table.

Operator	Meaning	Example
-b	file exists and is block special	[-b <FILE>]
-c	file exists and is character special	[-c <FILE>]
-d	file exists and is a directory	[-d <DIRECTORY>]
-e	file exists	[-e <FILE>]
-f	file is a regular file	[-f <FILE>]
-L	file exists and is a symbolic link	[-L <FILE>]
-r	file exists and read permission is granted	[-r <FILE>]
-s	file exists and has a size greater than zero	[-s <FILE>]
-w	file exists and write permission is granted	[-w <FILE>]
-x	file exists and execute (or search) permission is granted	[-x <FILE>]

Bash also offers a few binary comparison operators for performing file comparison. These operators are defined in the following table.

Operator	Meaning	Example
-ef	FILE1 has the same device and inode number as FILE2	[<FILE1> -ef <FILE2>]
-nt	FILE1 has newer modification date than FILE2	[<FILE1> -nt <FILE2>]
-ot	FILE1 has older modification date than FILE2	[<FILE1> -ot <FILE2>]

Important

The space characters on the inside of the brackets of the test expression, as well as those used to separate the elements within the test expression, are not there for readability, but rather are required for the proper evaluation of the expression. If any of these space characters are missing, the test will either fail or produce inaccurate or unexpected results.

Logical AND, OR operators

Situations may arise where it may be useful to test more than one condition. Bash's logical *AND* operator, **&&**, allows users to perform a compound condition test to see if both of two conditions are true. On the other hand, Bash's logical *OR* operator, **||**, allows users to test whether one of two conditions are true. The following examples demonstrate the use of Bash's logical AND and OR operators.

```
[student@server1 ~]$ [ 2 -gt 1 ] && [ 1 -gt 0 ]; echo $?
0
[student@server1 ~]$ [ 2 -gt 1 ] && [ 1 -gt 2 ]; echo $?
1
[student@server1 ~]$ [ 2 -gt 1 ] || [ 1 -gt 2 ]; echo $?
0
[student@server1 ~]$ [ 0 -gt 1 ] || [ 1 -gt 2 ]; echo $?
1
```

Using conditional structures

Simple shell scripts represent a collection of commands which are executed from beginning to end. Conditional structures allow users to incorporate decision making into shell scripts, so that certain portions of the script are executed only when certain conditions are met.

If/then statement

The simplest of the conditional structures in Bash is the if/then construct, which has the following syntax.

```
if <CONDITION>; then
      <STATEMENT>
      ...
      <STATEMENT>
    fi
```

With this construct, if a given condition is met, one or more actions are taken. If the given condition is not met, then no action is taken. The numeric, string, and file tests previously demonstrated are frequently utilized for testing the conditions in **`if/then`** statements. The following code section demonstrates the use of an **`if/then`** statement to start the *psacct* service if it is not active.

```
systemctl is-active psacct > /dev/null 2>&1

if  [ $? -ne 0 ]; then
  systemctl start psacct
fi
```

If/then/else statement

The **`if/then`** conditional structure can be further expanded so that different sets of actions can be taken depending on whether a condition is met. This is accomplished with the **`if/then/else`** conditional construct.

```
if <CONDITION>; then
      <STATEMENT>
      ...
      <STATEMENT>
    else
      <STATEMENT>
      ...
      <STATEMENT>
fi
```

The following code section demonstrates the use of an **`if/then/else`** statement to start the *psacct* service if it is not active and to stop it if it is active.

```
systemctl is-active psacct > /dev/null 2>&1

if  [ $? -ne 0 ]; then
  systemctl start psacct
else
  systemctl stop psacct
fi
```

If/then/elif/then/else statement

Lastly, the **`if/then/else`** conditional structure can be further expanded to test more than one condition, executing a different set of actions when a condition is met. The construct for this is shown in the following example. In this conditional structure, Bash will test the conditions in the order presented. Upon finding a condition that is true, Bash will execute the actions associated with the condition and then skip the remainder of the conditional structure. If none of the conditions are true, then Bash will execute the actions enumerated in the **`else`** clause.

```
if <CONDITION>; then
      <STATEMENT>
      ...
      <STATEMENT>
    elif <CONDITION>; then
      <STATEMENT>
      ...
      <STATEMENT>
    else
      <STATEMENT>
      ...
      <STATEMENT>
    fi
```

The following code section demonstrates the use of an **`if/then/elif/then/else`** statement to run the **`mysql`** client if the *mariadb* service is active, run the **`psql`** client if the *postgresql* service is active, or run the **`sqlite3`** client if both the *mariadb* and *postgresql* services are not active.

```
systemctl is-active mariadb > /dev/null 2>&1
MARIADB_ACTIVE=$?
systemctl is-active postgresql > /dev/null 2>&1
POSTGRESQL_ACTIVE=$?
```

```
if  [ "$MARIADB_ACTIVE" -eq 0 ]; then
  mysql
elif  [ "$POSTGRESQL_ACTIVE" -eq 0 ]; then
  psql
else
  sqlite3
fi
```

Case statement

Users can add as many **elif** clauses as they want into an **if/then/elif/then/else** statement to test as many conditions as they need. However, as more are added, the statement and its logic becomes increasingly harder to read and comprehend. For these more complex situations, Bash offers another conditional structure known as *case statements*. The **case** statement utilizes the following syntax:

```
case <VALUE> in
   <PATTERN1>)
       <STATEMENT>
       ...
       <STATEMENT>
       ;;
   <PATTERN2>)
       <STATEMENT>
       ...
       <STATEMENT>
       ;;
esac
```

The **case** statement attempts to match **<VALUE>** to each **<PATTERN>** in order, one by one. When a pattern matches, the code segment associated with that pattern is executed, with the **;;** syntax indicating the end of the block. All other patterns remaining in the **case** statement are then skipped and the **case** statement is exited. As many pattern/statement blocks as needed can be added.

To mimic the behavior of an **else** clause in an **if/then/elif/then/else** construct, simply use ***** as the final pattern in the **case** statement. Since this expression matches anything, it has the effect of executing a set of commands if none of the other patterns are matched.

The **case** statements are widely used in init scripts. The following code section is an example of how they are commonly used to take different actions, depending on the argument passed to the script.

```
case "$1" in
   start)
       start
       ;;
   stop)
       rm -f $lockfile
       stop
       ;;
   restart)
       restart
       ;;
   reload)
       reload
       ;;
   status)
```

```
        status
        ;;
    *)
        echo "Usage: $0 (start|stop|restart|reload|status)"
        ;;
esac
```

If the actions to be taken are the same for more than one pattern in a case statement, the patterns can be combined to share the same action block, as demonstrated in the following example. The pipe character, **|**, is used to separate the multiple patterns.

```
case "$1" in
   ...
   reload|restart)
       restart
       ;;
   ...
esac
```

References

bash(1) and **test**(1) man pages

Practice: Enhancing Bash Shell Scripts with Conditionals and Control Structures

Guided exercise

In this lab, you will create a Bash shell script to automate the process of creating Apache virtual hosts.

Resources:	
Machines:	**server1**

Outcomes:
A Bash script which automates the process of creating Apache virtual host configuration files and a document root directory. The script will perform the necessary checks to accommodate first-time execution of the script on a server, as well as ensure that virtual host conflicts do not occur.

Before you begin...
- Reset **server1**.
- Log into **server1** and become root with **su**.

Your company provides web hosting service to customers, and you have been tasked with writing a Bash shell script called **/usr/local/sbin/mkvhost** to automate the many steps involved in setting up an Apache name-based virtual host for your customers. The script will be used for virtual host creation on all servers going forward, so it needs to also be able to accommodate the one-time tasks that are executed the first time a new server is configured for name-based virtual hosting.

The script will take two arguments. The first argument will be the fully qualified domain name of the new virtual host. The second argument will be a number between 1 and 3, which represents the support tier that the customer purchased. The support tier determines the support email address, which will be set with the Apache **ServerAdmin** directive for the virtual host.

The script will create a configuration file under **/etc/httpd/conf.vhosts.d** with the name **<VIRTUALHOSTNAME>.conf** for each virtual host. It will also create a document root directory for the virtual host at **/srv/<VIRTUALHOSTNAME>/www**. Prior to creating the virtual host configuration file and document root directory, the script will check to make sure they do not already exist to ensure there will not be a conflict.

1. Install the *httpd* package, then enable and start **httpd**.

   1.1. Install the *httpd* package with **yum**.

   ```
   [root@server1 ~]# yum install -y httpd
   ```

   1.2. Enable and start **httpd**.

   ```
   [root@server1 ~]# systemctl enable httpd
   ln -s '/usr/lib/systemd/system/httpd.service' '/etc/systemd/system/multi-
   user.target.wants/httpd.service'
   ```

```
[root@server1 ~]# systemctl start httpd
```

2. Begin writing your script. Store the first and second argument of the script in the **VHOSTNAME** and **TIER** variables, respectively. Set the following variables:

Variable	Value
HTTPDCONF	/etc/httpd/conf/httpd.conf
VHOSTCONFDIR	/etc/httpd/conf.vhosts.d
DEFVHOSTCONFFILE	$VHOSTCONFDIR/00-default-vhost.conf
VHOSTCONFFILE	$VHOSTCONFDIR/$VHOSTNAME.conf
WWWROOT	/srv
DEFVHOSTDOCROOT	$WWWROOT/default/www
VHOSTDOCROOT	$WWWROOT/$VHOSTNAME/www

2.1. Create the new script file with a text editor.

```
[root@server1 ~]# vim /usr/local/sbin/mkvhost
```

2.2. Specify the interpreter program for the script.

```
#!/bin/bash
```

2.3. Set the variables for the arguments.

```
# Variables
VHOSTNAME=$1
TIER=$2
```

2.4. Set the other variables.

```
HTTPDCONF=/etc/httpd/conf/httpd.conf
VHOSTCONFDIR=/etc/httpd/conf.vhosts.d
DEFVHOSTCONFFILE=$VHOSTCONFDIR/00-default-vhost.conf
VHOSTCONFFILE=$VHOSTCONFDIR/$VHOSTNAME.conf
WWWROOT=/srv
DEFVHOSTDOCROOT=$WWWROOT/default/www
VHOSTDOCROOT=$WWWROOT/$VHOSTNAME/www
```

3. Check the argument values in the **VHOSTNAME** and **TIER** variables. If either is blank, display the message *"Usage: mkvhost VHOSTNAME TIER"* and exit with a status of **1**. If the arguments are passed correctly, then use a case statement to set a **VHOSTADMIN** variable to the proper support email address, based on the value of **$TIER**. The case statement will use the **$TIER** values of **1**, **2**, and **3** to set **VHOSTADMIN** to the corresponding support email address. If any other **$TIER** value is encountered, the case statement should display the message *"Invalid tier specified."* and exit with a status of **1**.

TIER	$VHOSTADMIN
1	basic_support@example.com

TIER	$VHOSTADMIN
2	business_support@example.com
3	enterprise_support@example.com

3.1. Create the if/then/else/fi statement. Use an OR conditional to check whether either of the arguments is blank and, if so, display the usage message and exit with a status of 1.

```
# Check arguments
if [ "$VHOSTNAME"  = '' ] || [ "$TIER" = '' ]; then
        echo "Usage: $0 VHOSTNAME TIER"
        exit 1
else
```

3.2. Create the case statement.

```
# Set support email address
        case $TIER in
                1)      VHOSTADMIN='basic_support@example.com'
                        ;;
                2)      VHOSTADMIN='business_support@example.com'
                        ;;
                3)      VHOSTADMIN='enterprise_support@example.com'
                        ;;
                *)      echo "Invalid tier specified."
                        exit 1
                        ;;
        esac
```

3.3. Close the if statement.

```
fi
```

4. Check to see if the **$VHOSTCONFDIR** directory is nonexistent. If so, create the directory. Check the exit status of the directory creation and display the error message *"ERROR: Failed creating $VHOSTCONFDIR"* if the directory creation failed.

```
# Create conf directory one time if non-existent
if [ ! -d $VHOSTCONFDIR ]; then
        mkdir $VHOSTCONFDIR

        if [ $? -ne 0 ]; then
                echo "ERROR: Failed creating $VHOSTCONFDIR."
                exit 1 # exit 1
        fi
fi
```

5. For Apache to be aware of the **$VHOSTCONFDIR** directory, you must have an include statement in **$HTTPDCONF**. Check to see if the following entry exists in the configuration file. If not, make a backup of the file to **$HTTPDCONF.orig** and then append the entry to the end of the configuration file. Check the exit status of the file modification and display the error message *"ERROR: Failed adding include directive"* if it failed.

```
# Add include one time if missing
```

```
grep -q '^IncludeOptional conf\.vhosts\.d/\*\.conf$' $HTTPDCONF

if [ $? -ne 0 ]; then
        # Backup before modifying
        cp -a $HTTPDCONF $HTTPDCONF.orig

        echo "IncludeOptional conf.vhosts.d/*.conf" >> $HTTPDCONF

        if [ $? -ne 0 ]; then
                echo "ERROR: Failed adding include directive."
                exit 1
        fi
fi
```

6. Check to see if a default virtual host already exists and, if not, create it.

   6.1. Verify if the default virtual host configuration file already exists and, if not, create and populate it with the following statement:

```
# Check for default virtual host
if [ ! -f $DEFVHOSTCONFFILE ]; then
        cat <<DEFCONFEOF > $DEFVHOSTCONFFILE
<VirtualHost _default_:80>
  DocumentRoot $DEFVHOSTDOCROOT
  CustomLog "logs/default-vhost.log" combined
</VirtualHost>

<Directory $DEFVHOSTDOCROOT>
  Require all granted
</Directory>
DEFCONFEOF
fi
```

   6.2. Verify if the default virtual host document root directory already exists and, if not, create it. Also set up the SELinux policy for the document root and apply it.

```
if [ ! -d $DEFVHOSTDOCROOT ]; then
        semanage fcontext -a -t httpd_sys_content_t '/srv(/.*)?'
        mkdir -p $DEFVHOSTDOCROOT
        restorecon -Rv /srv/
fi
```

7. Check to see if the virtual host's configuration file already exists and, if so, display the error message *"ERROR: $VHOSTCONFFILE already exists."* and exit with a status of **1**. Check to see if the virtual host's document root directory already exists and, if so, display the error message *"ERROR: $VHOSTDOCROOT already exists."* and exit with a status of **1**. If no errors are encountered with the previous two checks, continue with the creation of the virtual host configuration file, **$VHOSTCONFFILE,** and document root directory, **$VHOSTDOCROOT**. Populate the virtual host configuration file with the following statement:

```
# Check for virtual host conflict
if [ -f $VHOSTCONFFILE ]; then
        echo "ERROR: $VHOSTCONFFILE already exists."
        exit 1
elif [ -d $VHOSTDOCROOT ]; then
        echo "ERROR: $VHOSTDOCROOT already exists."
        exit 1
```

```
else
        cat <<CONFEOF > $VHOSTCONFFILE
<Directory $VHOSTDOCROOT>
  Require all granted
  AllowOverride None
</Directory>

<VirtualHost *:80>
  DocumentRoot $VHOSTDOCROOT
  ServerName $VHOSTNAME
  ServerAdmin $VHOSTADMIN
  ErrorLog "logs/${VHOSTNAME}_error_log"
  CustomLog "logs/${VHOSTNAME}_access_log" common
</VirtualHost>
CONFEOF

        mkdir -p $VHOSTDOCROOT
        restorecon -Rv $WWWROOT
fi
```

8. Verify the syntax of the configuration file with the command **`apachectl configtest`**. If there are no errors, then reload **`httpd.service`**. Otherwise, display the error message *"ERROR: Configuration error."* and exit with a status of **`1`**.

```

# Check config and reload
apachectl configtest &> /dev/null

if [ $? -eq 0 ]; then
        systemctl reload httpd &> /dev/null
else
        echo "ERROR: Config error."
        exit 1
fi
```

9. Save and execute the script.

   9.1. Make the script executable.

```
[root@server1 ~]# chmod u+x /usr/local/sbin/mkvhost
```

   9.2. Execute the script.

```
[root@server1 ~]# /usr/local/sbin/mkvhost www1.example.com 3
```

Summary

Enhancing Bash Shell Scripts with Conditionals and Control Structures

In this section, students learned how to:

- Extend script functionality with the use of positional parameters.
- Evaluate exit status to verify success or failure of program execution.
- Perform tests on script inputs, files, and directories with the **`test`** command.
- Utilize **`if/then`** and **`case`** statements to perform different actions for different conditions.

CHAPTER 12

CONFIGURING THE SHELL ENVIRONMENT

Overview	
Goal	To customize Bash startup and use environment variables, Bash aliases, and Bash functions.
Objectives	• Use bash startup scripts to define environment variables, aliases, and functions
Sections	• Changing the Shell Environment (and Practice)
Lab	• Configuring the Shell Environment

Changing the Shell Environment

Objectives

After completing this section, students should be able to describe environment variables, configure them in **bash** start up scripts, define **bash** aliases, and use functions.

Environment variables

The shell and scripts use variables to store data; some variables can be passed to sub-processes along with their content. These special variables are called *environment variables*.

Applications and sessions use these variables to determine their behavior. Some of them are likely familiar to administrators, such as **PATH**, **USER**, and **HOSTNAME**, among others. What makes variables environment variables is that they have been *exported* in the shell. A variable that is flagged as an export will be passed, with its value, to any sub-process spawned from the shell. Users can use the **env** command to view all environment variables that are defined in their shell.

Any variable defined in the shell can be an environment variable. The key to making a variable become an environment variable is flagging it for export using the **export** command.

In the following example, a variable, **MYVAR**, will be set. A sub-shell is spawned, and the **MYVAR** variable does not exist in the sub-shell.

```
[student@demo ~]$ MYVAR="some value"
[student@demo ~]$ echo $MYVAR
some value
[student@demo ~]$ bash
[student@demo ~]$ echo $MYVAR
[student@demo ~]$ exit
```

In a similar example, the **export** command will be used to tag the **MYVAR** variable as an environment variable, which will be passed to a sub-shell.

```
[student@demo ~]$ MYVAR="some value"
[student@demo ~]$ export MYVAR
[student@demo ~]$ echo $MYVAR
some value
[student@demo ~]$ bash
[student@demo ~]$ echo $MYVAR
some value
[student@demo ~]$ exit
```

bash start-up scripts

One place where environment variables are used is in initializing the **bash** environment upon user log in. When a user logs in, several shell scripts are executed to initialize their environment, starting with the **/etc/profile**, followed by a profile in the user's home directory, typically **~/.bash_profile**.

Note

A **bash** shell will look for one of three files in a user's home directory: **.bash_profile**, **.bash_login**, or **.profile**. The shell will look for the files in the given order, and will execute the first file it locates.

Because these profiles have additional scripting in them, which calls other shell scripts, the **bash** login scripting will typically be the following:

```
/etc/profile
           \__ /etc/profile.d/*.sh

~/.bash_profile
              \__ ~/.bashrc
                         \__ /etc/bashrc
```

Generally, there are two types of login scripts, profiles and "RCs". Profiles are for setting and exporting of environment variables, as well as running commands that should only be run upon login. RCs, such as **/etc/bashrc**, are for running commands, setting aliases, defining functions, and other settings that cannot be exported to sub-shells. Usually, profiles are only executed in a login shell, whereas RCs are executed every time a shell is created, login or non-login.

The layout of the file call is such that a user can override the default settings provided by the system wide scripts. Many of the configuration files provided by Red Hat will contain a comment indicating where user-specific changes should be added.

Using alias

alias is a way administrators or users can define their own command to the system or override the use of existing system commands. Aliases are parsed and substituted prior to the shell checking **PATH**. **alias** can also be used to display all existing aliases defined in the shell.

```
[student@demo ~]$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

All the default aliases defined in a user environment are in the previously shown output of **alias**. The definition of the **ll** alias means that when the user types **ll** as a command, the shell will expand the alias and execute **ls -l --color=auto**. In this way, a new command **ll** has been added to the shell. In another example, the alias for **grep**, the alias overrides the default invocation of an existing command on the system. When a user enters the **grep** command, the shell will expand the alias and substitute the **grep --color=auto** command. Due to the alias, all calls to **grep** are overridden to become calls to **grep** with the **--color** option passed automatically.

Use the **alias** command to set an alias. The defined alias will exist for the duration of the current shell only.

```
alias mycomm="<command to execute>"
```

```
[student@demo ~]$ alias usercmd='echo "Hurrah!"; ls -l'
[student@demo ~]$ usercmd
Hurrah!
total 0
-rw-rw-r--. 1 student student 0 Jun  9 13:21 file1
-rw-rw-r--. 1 student student 0 Jun  9 13:21 file2
-rw-rw-r--. 1 student student 0 Jun  9 13:21 file3
```

To make the alias persistent, the user would need to add the command to the bottom of their **~/.bashrc**.

```
[student@demo ~]$ vi ~/.bashrc
...
# User specific aliases and functions
alias usercmd='echo "Hurrah!"; ls -l'
```

After the **alias** is added to the **~/.bashrc**, it will be available in every shell created.

To remove an alias from the environment, use the **unalias** command.

Using functions

When used in shell scripts, functions are a way of isolating a segment of code so that it can be called repeatedly without having to write the entire segment again. Additionally, if the code requires an update, the function's content can be updated, and everywhere the function is referenced, the updated code is now executed.

An example of defining and using a function within a shell script follows, taken from **/etc/profile**. The **pathmunge** function takes two arguments; the first is a directory, the second (optional) is the word "after". Based on whether "after" is passed as $2, the directory will be added to the **PATH** environment variable at the front or end of the existing list of directories. Later, the function is invoked several times to build the **PATH** for root or regular users. Notice that for root, all the directories are prepended to **PATH**, where regular users have their **PATH** built by appending.

```
 pathmunge () {
  if [ "$2" = "after" ] ; then
    PATH=$PATH:$1
  else
    PATH=$1:$PATH
  fi
}
...
if [ "$EUID" = "0" ]; then
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /usr/local/sbin
else
    pathmunge /usr/local/sbin after
    pathmunge /usr/sbin after
    pathmunge /sbin after
fi
```

Functions can also be set in the **bash** shell environment. When set in the environment, they can be executed as commands on the command line, similar to aliases. Unlike aliases, they can take

arguments, be much more sophisticated in their actions, and provide a return code. Functions can be defined in the current shell by typing them into the command line, but more realistically, they should be set in a user's **~/.bashrc** or the global **/etc/bashrc**.

There are many functions set by default in the user environment, which can be viewed with the **set** command. **set** will display all functions and variables in the current shell environment. To remove a function from the environment, a user or administrator may use the **unset** command, passing the function or variable name to remove.

References

bash(1), **env**(1), and **builtins**(1) man pages

Practice: Working with Login and Non-Login Shells

Guided exercise

In this lab, you will apply changes to the scripts used to establish the initial shell environment for log in and non-login shells.

Resources:	
Files:	• /etc/profile • /etc/bashrc • ~/.bash_profile • ~/.bashrc
Machines:	server1

Outcomes:
Shells will now have additional environment variables, functions, and aliases defined.

Before you begin...
Reset your **server1** system. After the reset completes, log in as the **student** user.

1. Change the student user's **PS1** environment variable to **[\u@\h \t \w]$** .

   1.1. Since this is an environment variable, and it should affect only student, edit **~/.bash_profile**.

   ```
   [student@server1 ~]$ vi ~/.bash_profile
   ```

   1.2. Add an entry for the **PS1** variable, setting it to **[\u@\h \t \w]$** .

   ```
    ...
   # User specific environment and startup programs

   PATH=$PATH:$HOME/.local/bin:$HOME/bin
   PS1='[\u@\h \t \w]$ '

   export PATH PS1
   ```

2. Set aliases for the student user so that when the **rm**, **cp**, or **mv** commands are used, they are automatically called with the **-i** option.

   2.1. Edit the student's **~/.bashrc** configuration file.

   ```
   [student@server1 ~]$ vi ~/.bashrc
   ```

   2.2. Add aliases for the commands into the file.

```
 ...
# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

3. You have decided that users on server1 need to have an application to help them divine the future. Add a function called **8ball**, available to all users, that selects a random number between zero and three. Based on the random value, the function should present a message to help them make a decision about the future.

   3.1. Edit the **/etc/bashrc** file as root.

```
[student@server1 ~]$ su
[root@server1 ~]# vi /etc/bashrc
```

   3.2. At the bottom of the file, add the **8ball** function.

```
 ...
8ball () {
  echo "Shaking ..."
  echo
  sleep 3
  value="$[ $RANDOM % 3 ]"
  case $value in
    0) echo "All signs point to yes." ;;
    1) echo "The answer is no." ;;
    2) echo "Ask again later." ;;
    3) echo "Outlook hazy." ;;
  esac
  echo
}
```

4. Use an **ssh** session to create a login shell to verify that the environment changes are effective.

```
[root@server1 ~]# exit
[student@server1 ~]$ ssh student@localhost
student@localhost's password: r3dh@t1!
[student@server1 00:06:57 ~]$ touch file1
[student@server1 00:06:57 ~]$ rm file1
rm: remove regular empty file ‘file1’? y
[student@server1 00:06:57 ~]$ 8ball
The answer is no.
```

Lab: Configuring the Shell Environment

Performance checklist

In this lab, you will make configuration changes to the shell environment for individual users, and all users, on the machine.

Your international coworkers have complained that when they log into the console of server1, the language on the machine is set incorrectly. You will configure the machine such that when a user logs in with the terminal type of **linux**, the **LANG** variable will be set to **en_US**.

The student user requires a command be added to their environment. **diskcheck** will run **iostat -d** and **df -hP --type xfs**.

The student user needs an environment variable **JAVA_HOME** set to **/usr/lib/jvm**.

Resources:	
Files:	• /etc/profile • ~/.bash_profile • ~/.bashrc
Machines:	server1

Outcomes:
All users will have their **LANG** variable set to **en_US** when using a **linux** terminal type. The student user will have a function, named **diskcheck**, defined in the environment. The student will have an environment variable, JAVA_HOME, defined in the environment.

Before you begin...
Reset your **server1** system.

1. Detect the terminal type of a shell. If it is **xterm**, make sure the language is set to **en_US**.

2. Add a function to the student user's environment. The function is called **diskcheck**, and when called, will display the output of the **iostat -d** and **df -hP --type xfs** commands.

```
[student@server1 ~]$ vi ~/.bashrc
```

3. Set an environment variable, JAVA_HOME, to **/usr/lib/jvm** for the student user.

```
[student@server1 ~]$ vi ~/.bash_profile
```

Solution

In this lab, you will make configuration changes to the shell environment for individual users, and all users, on the machine.

Your international coworkers have complained that when they log into the console of server1, the language on the machine is set incorrectly. You will configure the machine such that when a user logs in with the terminal type of **linux**, the **LANG** variable will be set to **en_US**.

The student user requires a command be added to their environment. **diskcheck** will run **iostat -d** and **df -hP --type xfs**.

The student user needs an environment variable **JAVA_HOME** set to **/usr/lib/jvm**.

Resources:	
Files:	• /etc/profile • ~/.bash_profile • ~/.bashrc
Machines:	server1

Outcomes:
All users will have their **LANG** variable set to **en_US** when using a **linux** terminal type. The student user will have a function, named **diskcheck**, defined in the environment. The student will have an environment variable, JAVA_HOME, defined in the environment.

Before you begin...
Reset your **server1** system.

1. Detect the terminal type of a shell. If it is **xterm**, make sure the language is set to **en_US**.

   Edit the **/etc/profile** file. At the bottom of the file, add a bit of script which will check the **TERM** variable to see if it is set to **xterm**. If it is, assign the **LANG** variable to **en_US** and **export** the **LANG** setting.

   ```
   [root@server1 ~]$ vi /etc/profile
   ```

   ```
   if [ "$TERM" == "xterm" ]
   then
     LANG=en_US
     export LANG
   fi
   ```

   To verify the setting works, ssh in to localhost as student. Check the value of the **LANG** variable.

   ```
   [root@server1 ~]# ssh student@localhost
   [student@server1 ~]$ echo $LANG
   en_US
   ```

2. Add a function to the student user's environment. The function is called **diskcheck**, and when called, will display the output of the **iostat -d** and **df -hP --type xfs** commands.

   Edit the student user's **~/.bashrc** and add the function definition. Add the new function at the end of the file.

```
[student@server1 ~]$ vi ~/.bashrc
```

```
# User specific aliases and functions
diskcheck() {
  iostat -d
  echo
  df -hP --type xfs
}
```

   Source the **~/.bashrc** and verify the function.

```
[student@server1 ~]$ . ~/.bashrc
[student@server1 ~]$ diskcheck
```

3. Set an environment variable, JAVA_HOME, to **/usr/lib/jvm** for the student user.

   Edit the student user's **.bash_profile**, and at the bottom of the file, add an entry for **JAVA_HOME**. Set **JAVA_HOME** to **/usr/lib/jvm**. Use the **export** command to tag the variable to be available for all sub-shells.

```
[student@server1 ~]$ vi ~/.bash_profile
```

```
JAVA_HOME=/usr/lib/jvm
export JAVA_HOME
```

   Source the **.bash_profile** to read the changes into the environment.

```
[student@server1 ~]$ . ~/.bash_profile
```

   Ensure the variable is available in sub-shells.

```
[student@server1 ~]$ bash
[student@server1 ~]$ echo $JAVA_HOME
/usr/lib/jvm
```

Summary

Changing the Shell Environment

In this section, students learned how to configure **bash** start up scripts to define environment variables, aliases, and functions.