# Red Hat System Administration III | RH254

# Quick Guide

**Version 24.05**

**Ahmed Abdelwahed**
**ahmed@abdelwahed.me**
**LinkedIn**

# Control services and daemons

`systemd` is a system and service manager for Linux operating systems that has become the standard for modern Linux distributions. It provides an array of system components for managing boot processes, system services, and more.

**Main Components**

- `Unit`: The basic building block of **systemd** service management. A unit may be a service, a mount point, a socket, etc., each defined with a unit file specifying how to manage and run the unit.
- `systemctl`: The primary command-line tool used to interact with **systemd**. It allows you to start, stop, restart, enable, and disable unit files.
- `journalctl`: A tool to inspect and manipulate logs created by **journald**, enabling system administrators to query and display the content of the systemd journal.
- `systemd-analyze`: A tool used to analyze system boot-up performance.

## View and Query Services

- `systemd-analyze` Analyze Boot Performance.
- `systemctl -t help` Shows the available units.
- `systemctl list-units` Shows all loaded units.
- `systemctl status sshd.service` Check service state.
- `systemctl --type=service` Check all active services.
- `systemctl --type=service –all` check all services regardless of their state.
- `systemctl list-units --type=service --state=running | wc -l`
- `systemctl --type=service | grep active | wc -l` Show total number of active processes.
- `systemctl is-active sshd` Check if sshd is active.
- `systemctl is-enabled sshd Check` if sshd is enabled.
- `systemctl --failed --type=service` List failed services.
- `journalctl -u sshd` View Service Logs

## Start, Stop, Reload Services

- `systemctl stop sshd.service` Stop sshd service.
- `systemctl start sshd.service` Start sshd service.
- `systemctl restart sshd.service` Restart sshd service.
- `systemctl reload sshd.service` Reload sshd service.
- `systemctl reload-or-restart sshd.service` Reload if available or restart sshd service.

## View Dependencies

- `systemctl list-dependencies sshd.service` Display dependencies hierarchy.

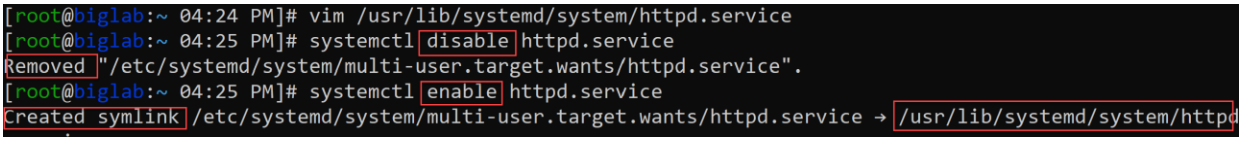
## Masking Services

- `systemctl mask name.service` Mask service to prevent it from being started.
- `systemctl unmask name.service` Unmask a service.
- `systemctl list-unit-files --state=masked` Get all masked services.

---

## Edit and Customize Services

- `/usr/lib/systemd/system/` Directory containing systemd service files.




- `vim /usr/lib/systemd/system/httpd.service` Edit the httpd service file.
- `systemctl cat httpd` Shows the configuration file for the httpd service.
- `systemctl show httpd` Display properties of httpd.
- `systemctl edit httpd` Edit the httpd service in the default editor.
- `systemctl daemon-reload` Reload system manager configuration.
- `systemctl restart httpd` Restart the httpd service.
- `ps aux | grep http` Find and kill the httpd process by PID, used in conjunction with killing the process to observe its state.

### Mount units in systemd

.mount units in systemd are used to manage filesystem mounts in a consistent and integrated manner with the system's initialization process. These units allow systemd to mount filesystems at boot time or on-demand when accessed, as well as to keep track of mounted filesystems and their options. Here are examples to illustrate how to create and manage .mount units.

Example : Creating a .mount Unit for a Data Partition

Suppose you have a data partition on /dev/sdb1 that you want to mount to /mnt/data.

1. **Create the .mount Unit File**: The name of the unit file must match the mount point's path, with slashes replaced by dashes (excluding the leading slash). For /data, the unit file name will be data.mount.
   - Create the file `/etc/systemd/system/mnt-data.mount` with the following content:

```
[Unit]
Description=Data Partition

[Mount]
What=/dev/sdb1
Where=/mnt/data
Type=ext4
Options=defaults

[Install]
WantedBy=multi-user.target
```

2. **Enable, Start and verify the Mount**:
   - First, reload the systemd configuration to recognize the new .mount unit:

```
systemctl daemon-reload

systemctl enable mnt-data.mount

systemctl start mnt-data.mount

df -h /mnt/data

systemctl list-dependencies mnt-sdb1.mount and systemctl list-units --type=mount
```

---

---

## Swap units in system

.swap units in systemd are used to manage swap spaces on a Linux system. Swap spaces are crucial for extending the virtual memory of a system by providing a space on disk where the kernel can temporarily store data that doesn't fit into physical RAM. Managing swap with systemd integrates swap space management into the system's initialization and service management framework, offering more flexibility and control.

Example : Creating a .swap Unit for a Swap File

Suppose you want to create and use a swap file located at /swapfile.

1. **Create the Swap File**:
   - First, create a swap file of the desired size. Here's how to create a 1GB swap file:

```
fallocate -l 2G /mnt/swapfile
chmod 600 /mnt/swapfile
mkswap /mnt/swapfile
```

2. **Create the .swap Unit File**:
   - The name of the .swap unit should correspond to the swap file path, with slashes replaced by dashes (except the leading slash). For a swap file at /swapfile, the unit file is named mnt-swapfile.swap.
   - Create the unit file `/etc/systemd/system/mnt-swapfile.swap` with the following content:

```
[Unit]
Description=Swap File
[Swap]
What=/mnt/swapfile
[Install]
WantedBy=multi-user.target
```

3. **Enable, Start and verify the Swap**:
   - Reload systemd to recognize the new .swap unit, enable it to activate at boot, and start it immediately:

```
systemctl daemon-reload
systemctl enable --now mnt-swapfile.swap
swapon --show
systemctl list-units --type=swap
systemctl list-dependencies mnt-swap1.swap and ll
/etc/systemd/system/mnt-swapfile1.swap
```

---

# What is a Shell?

A **shell** is a user interface for accessing an operating system's services. It can be graphical (GUI) or command-line (CLI). In the context of Unix and Linux systems, when we talk about "the shell," we're often referring to a command-line interface that allows users to interact directly with the operating system through typed commands.
Shells enable users to launch programs, manage files and directories, and perform a variety of other tasks by typing commands into a terminal or console window. They can also control the operating system kernel directly through system calls or indirectly by executing utilities.

Types of Shells in Linux:

- **Bash (Bourne Again SHell)**: The most common Linux shell. It is an improvement of the original Bourne shell (`sh`), adding additional features such as command completion and command history.
- **Tcsh and Csh (C Shell)**: Shells with syntaxes resembling the C programming language, known for their scripting capabilities and interactive use.
- **Zsh (Z Shell)**: Combines features of Bash, ksh, and tcsh, with themes and plugins, offering extensive customization.
- **Fish (Friendly Interactive SHell)**: Known for its user-friendly features, like autosuggestions and a web-based configuration interface.

Core Functions of a Shell:

- **Command Interpretation**: It interprets the commands that users type into the terminal or console and then executes them.
- **Script Execution**: Shells can execute scripts, which are files containing a series of commands, allowing for automation of tasks.
- **Environment Management**: Shells manage the user environment, including the setting of environment variables which can control the behavior of certain programs.
- **Job Control**: Most shells allow users to control the execution of jobs (programs, scripts, or commands). This includes starting, stopping, and resuming jobs.

# Shell Scripting

Shell scripting is a method to automate repetitive tasks by creating a series of commands in a script file that can be executed by the shell (command interpreter).

## Basics of Writing Shell Scripts

- **Creating a Script**: A shell script typically starts with a "shebang" (`#!`) followed by the path to the interpreter that should execute the script (e.g., `#!/bin/bash` for Bash scripts).
- **Writing Commands**: Scripts can contain any commands you might type at the command line. These are executed in sequence.
- **Making Scripts Executable**: After writing a script, you must make it executable using the `chmod` command, e.g., `chmod +x myscript.sh.`
- **Running a Script**: Execute a script by typing its path in the shell, like `./myscript.sh`, assuming you're in the same directory.
- **Profiles and Configuration Files**: Understanding how to set and use environment variables through files like `.bashrc`, `.profile`, or `/etc/profile` for global variables is crucial for scripting. To print all the environment variables along with their values use: `printenv`, `env`, `set`
- **Quoting:** Use quotes to handle strings and variables that include spaces and special characters.

---

- **Exit Statuses:** Scripts should handle exit statuses of commands `(echo $?)` to check if a command succeeded or failed, where 0 for successful and Non-zero exit statuses indicate different types of errors or statuses

## Understanding the set Command in Bash

The `set` command in Bash is a powerful built-in utility that controls the behavior of the shell itself and is often used to define how scripts handle certain situations like error handling and argument processing.

**1. `set --`: Resetting Positional Parameters**

One of the uses of the `set` command is to reset all positional parameters using `set --`. This is particularly useful when you need to clear existing positional parameters or set new ones explicitly.

- **Usage**: `set -- "value1" "value2"`
- **Example**: Resetting parameters for a script.

```
# Before
echo $1 $2  # Outputs current $1 and $2
set -- "Hello" "World"
# After
echo $1 $2  # Outputs: Hello World
```

In this example, the positional parameters `$1` and `$2` are reset to "Hello" and "World" respectively. It can be beneficial when parameters need to be redefined due to changing input requirements within a script.

## 2. `set -e`: Error Handling in Scripts

Another critical use of the `set` command is `set -e`, which tells the shell to exit immediately if a command within a script exits with a non-zero status (indicating failure).

- **Usage**: `set -e`

```
#!/bin/bash
set -e  # Exit on error

echo "Starting the script..."
# If the next command fails, the script will exit immediately.
cp important_file backup_file

echo "Script completed successfully.
```

**Install and Configure Nginx Homepage**

This script installs Nginx, starts and enables its service, adds firewall rules, and creates a welcome page.

```
# Install Nginx
      yum install -y nginx
# Enable and start Nginx service
      systemctl enable nginx
      systemctl start nginx
# Configure firewall rules
      firewall-cmd --add-port=80/tcp --permanent
      firewall-cmd --add-port=80/udp --permanent
      firewall-cmd --reload
# Create a welcome page
      echo "Welcome to Oman" > /usr/share/nginx/html/index.html
```

---

---

# Functions

## 1. big_files()

**Purpose**: Finds and lists the top 10 largest files larger than a specified size within a given directory, displaying them in descending order of size.

**Full Function**:

```
big_files() {
    local directory=$1
    local min_size=$2
    # Find files larger than $min_size MB in $directory, sort them by size
    find "$directory" -type f -size +"${min_size}M" -exec du -h {} + | sort -rh | head -n 10
}
```

## 2. monitor_log()

**Purpose**: Continuously monitors a log file for new entries containing keywords like "ERROR", "FAIL", or "WARNING" and highlights them.

**Full Function**:

```
monitor_log() {
    local logfile=$1
    # Tail the logfile and highlight lines containing ERROR, FAIL, or WARNING
    tail -f "$logfile" | grep --color -iE "ERROR|FAIL|WARNING"
}
```

## 3. disk_usage()

**Purpose**: Displays the disk usage of all mounted filesystems, excluding temporary filesystems and CD-ROM drives, sorted by usage percentage in descending order.

**Full Function**:

```
disk_usage() {
    # Display disk usage, exclude non-essential filesystems like tmpfs, cdrom, and sr, then sort
by usage percentage
    df -h | egrep -v '(tmpfs|cdrom|sr)' | awk '{print $5 " " $1}' | sort -rh
}
```

## 4. check_port()

**Purpose**: Checks if a port on a specified host is open and prints the status, enhancing network troubleshooting.

**Full Function**:

```
check_port() {
    local host=$1
    local port=$2
    # Check if the port at the host is open and print the status
    if nc -zv "$host" "$port"; then
        echo "Port $port on $host is open."
    else
        echo "Port $port on $host is closed."
    fi
}
```

---

---

## 5. backup_file()

**Purpose**: Creates a timestamped backup of a specified file, storing it in a predefined backup directory to prevent data loss.
**Full Function**:

```
backup_file() {
    local file_path=$1
    local backup_dir="/bk"
    local timestamp=$(date +%Y%m%d%H%M%S)
    local backup_path="${backup_dir}/$(basename "$file_path").${timestamp}.bak"

    # Ensure the backup directory exists
    mkdir -p "$backup_dir"

    # Copy the file to the backup directory with a timestamp
    if cp -v "$file_path" "$backup_path"; then
        echo "Backup of $file_path created at $backup_path"
    else
        echo "Failed to create backup of $file_path"
    fi
}
```

---

# MANAGING NETWORK

## Network Management with nmcli

- **Show Devices** `nmcli d s` or `ip link` or `nmcli d`
- **Show Connections** `nmcli connection show` or `nmcli c`
- **Device Status** `nmcli dev status`
- **Add a Connection to NIC (Static)**
  ```
  nmcli connection add con-name newcon1 ifname ens224 ipv4.addresses 192.168.219.140/24
  ipv4.gateway 192.168.219.2 ipv4.dns 1.1.1.1 ipv4.method manual type ethernet
  ```
- **Modify an Existing Connection**
- `nmcli connection modify "ens160" ipv4.addresses "192.168.219.111/24" ipv4.gateway "192.168.219.2" ipv4.dns "1.1.1.1" ipv4.dns-search "search.abdelwahed.me" ipv4.method manual autoconnect yes`
- **Adding secondary DNS** `nmcli con mod newcon1 +ipv4.dns 8.8.8.8` The **+** sign indicates that you're adding to the existing list of DNS servers rather than replacing it. If you wanted to replace the entire list, you would use **ipv4.dns** without the plus sign.
- **Add extra connection to the same NIC (DHCP)**
  `nmcli con add type ethernet con-name newcon2 ifname ens224 ipv4.method auto`
- **Reload Connection** `nmcli connection reload`
- **Activate a Connection** `nmcli connection up "newcon1"`
- **Show the details of a network connection** nmcli con show newcon1
- **Deactivate a Connection** `nmcli connection down newcon1`
- **Delete Connection** `nmcli con del newcon2`
- **Disable NIC** `ip link set dev ens224 down`
- `ss -t -a && netstat -tuln`
- `yum install nmap`
- `nmap -v -A 192.168.244.162`
- `tcpdump -i team0 -w packet_capture.pcap`
- `tcpdump -r packet_capture.pcap`
- **nm-connection-editor**
  The `nm-connection-editor` is a graphical utility for managing network connections on systems that use NetworkManager. It provides an easy-to-use interface for creating, editing, and deleting wired, wireless, VPN, and other types of network connections. If you're encountering issues with network configurations or just prefer a GUI over command-line tools, nm-connection-editor can be a helpful alternative.

## Modifying the system hostname

- `sudo hostnamectl set-hostname server01 --static`
- `sudo hostnamectl set-hostname server01-dynamic --transient`
- `sudo hostnamectl set-hostname " Family Media Center 📽" –pretty`
- **Remove Pretty Hostname** `sudo hostnamectl set-hostname "" –prettyZ`

# Network Teaming

## Add New Team

**Objective:**

- Combine two or more network interfaces into a single virtual interface to provide network redundancy and increased throughput.

**Prerequisites:**

- RHEL installed on a system or VM with at least two network interfaces.
- Root or sudo privileges.

**Step 1: Install the Teaming Driver**

1. Open a terminal.
2. Update your system: `sudo dnf update`
3. Install the teaming driver and utilities: `sudo dnf -y install teamd libteam`

**Step 2: Identify Network Interfaces**

1. Identify the names of the network interfaces you wish to team: `nmcli dev status`
2. Note down the interface names, e.g., `ens160, ens224.`

**Step 3: Create a Team Interface**

1. Bring down the connections and interfaces:
   - `nmcli c delete ens160 && nmcli c delete ens224`
   - `nmcli dev dis ens160` "if needed"
   - `nmcli dev dis ens224` "if needed"
2. Create the team interface: `nmcli con add type team con-name team0 ifname team0`
3. Configure the team: `nmcli con mod team0 team.config '{"runner":{"name":"activebackup"}}'`
   - This example uses the `activebackup` runner for redundancy. Other options include `loadbalance`, `roundrobin`, etc., based on your requirements.

**Step 4: Add Interfaces to the Team**

1. Add the first interface to the team:
   - `nmcli con add type team-slave con-name team0-port1 ifname ens160 master team0`
2. Add the second interface to the team:
   - `nmcli con add type team-slave con-name team0-port2 ifname ens224 master team0`

**Step 5: Bring up the Team Interface**

1. Enable the team interface: `nmcli con up team0`
2. Verify the teaming configuration: `ip a s` and `teamdctl team0 state`

**Step 6: Configure IP Addressing**

- Assign an IP address to the team interface (example uses DHCP): `sudo nmcli con mod team0 ipv4.method auto`
- Alternatively, assign a static IP address: `sudo nmcli con mod team0 ipv4.addresses 192.168.244.99/24 ipv4.gateway 192.168.244.2 ipv4.dns "1.1.1.1 8.8.8.8 " ipv4.method manual autoconnect yes`
- Reactivate the team interface for changes to take effect: `sudo nmcli con up team0`

**Step 7: Test and Verify**

- **Ensure all interfaces are up:** `nmcli con show`
- **Test network connectivity:** `ping -c 4 google.com`
- **Check the team interface status again**: `teamdctl team0 state`
- **Check the speed of a network card:** `ethtool team0`

---

## Add the New NIC to the Team

Now, add the new NIC to team0 by creating a new team-slave connection.

- `nmcli con add type team-slave con-name team0-port3 ifname ens256 master team0`

**Bring Up the New Team-Slave Connection**

Activate the new team-slave connection to make the new NIC an active part of team0:

- `nmcli con up team0`

**Verify the Team Configuration**

Finally, check the status of the team to verify that the new NIC has been successfully added and is active within the team:

- `teamdctl team0 state`

## Change the Runner Configuration

**Bring Down the Team Interface**

First, bring down the team interface to ensure that changes can be applied smoothly. This step prevents any potential disruption in network connectivity due to the configuration change.

- `nmcli con down team0`

**Modify the Runner Configuration**

Next, modify the team configuration to change the runner. In this example, let's change the runner to loadbalance, which balances outgoing traffic across the active ports based on a hash of the packet header:

- `nmcli con mod team0 team.config '{"runner":{"name":"loadbalance"}}'`

Replace "loadbalance" with the name of the runner you wish to use. Each runner has its own set of options and configurations, so you might need to consult the documentation for specific runner configurations.

**Bring Up the Team Interface**

After modifying the runner, bring the team interface back up to apply the changes:

- `nmcli con up team0`

**Verify the Changes**

Finally, verify that the changes have been applied correctly by examining the team configuration:

- `teamdctl team0 state`

## Delete Team0

**Step 1: Bring Down the Team Interface**

First, disable the team interface to ensure a smooth deletion process.

- `nmcli con down team0`

**Step 2: Delete the Team-Slave Interfaces**

Before deleting the team itself, remove the slave interfaces associated with it. This will prevent any potential conflicts or leftover configurations.

- `nmcli con delete team0-port1`
- `nmcli con delete team0-port2`

**Note:** The actual names of the slave connections (team-slave-ens160 and team-slave-ens224) might differ, based on how they were originally named when you added them to the team. If you're unsure about the names, you can list all connections with nmcli con show and identify the team-slave connections associated with ens160 and ens224.

**Step 3: Delete the Team Interface**

Now that the slave interfaces are removed, you can safely delete the team interface itself.

- `nmcli con delete team0`

**Step 4: Verify the Deletion**

Ensure that the team and its components have been completely removed by listing all network connections:

- `nmcli con show`

---

# Managing Firewall with Firewalld

Firewalld is a dynamic firewall management tool available on many Linux distributions. It provides a way to configure and manage network firewalls, including creating, modifying, and deleting firewall rules.

## Firewalld Configuration and Management

1. **Firewall Configuration File**
   - Location `/etc/firewalld/firewalld.conf`
   - default zone XML files directory `/usr/lib/firewalld/zones/`
   - `cat /usr/lib/firewalld/zones/block.xml`
   - `cat /usr/lib/firewalld/zones/public.xml`
   - Service definitions are stored in `/usr/lib/firewalld/services`
   - `firewall-cmd --add-service=testservice --permanent`
2. **Checking Firewalld Service Status**
   - `systemctl status firewalld.service` or `sudo firewall-cmd --state` View the status of the Firewalld service.
3. **Enabling/Starting/Stopping Firewalld Service**
   - `systemctl enable firewalld.service` Enable Firewalld to start at boot.
   - `systemctl start firewalld.service` Start Firewalld service.
   - `systemctl stop firewalld.service` Stop Firewalld service.
4. **Managing Firewall with firewall-cmd**
   - `firewall-cmd` Primary command to manage the firewall.
   - `firewall-cmd --list-all` List all current settings.
5. **Working with Zones**
   - List available zones `firewall-cmd --get-zones`
   - List all settings for all zones `firewall-cmd --list-all-zones`
   - Get default zone `firewall-cmd --get-default-zone`
   - Display active zones `firewall-cmd --get-active-zones`
   - Set default zone `firewall-cmd --set-default-zone=internal`
   - Add service to a zone `firewall-cmd --zone=public --add-service=http`
   - List services in a zone `firewall-cmd --zone=public --list-services`
   - Assign a network interface to a specific zone `firewall-cmd --zone=work --change-interface=ens160 --permanent`
   - Remove NIC from public zone `firewall-cmd --permanent --zone=public --remove-interface=ens160`
   - `firewall-cmd --get-zone-of-interface=ens160`
   - Get all details about a particular zone `firewall-cmd --zone=public --list-all`
   - Check service status `firewall-cmd --query-service=ssh`
   - Check port status `firewall-cmd --query-port=8080/tcp`
6. **Managing Firewall Services and Ports**
   - Remove a service `firewall-cmd --remove-service=ssh`
   - Add a service temporarily `firewall-cmd --add-service=ssh`
   - Add a service permanently `firewall-cmd --add-service=ssh –permanent`
   - List all allowed services `firewall-cmd --list-services`
   - List all allowed services in public zone `firewall-cmd --zone=public --list-services`

---

- List all predefined services that come with firewalld `firewall-cmd --get-services`
- Add a port permanently `firewall-cmd --add-port=5050/tcp --permanent`
- List all open ports `firewall-cmd --list-ports`
- `ss -nltpn`
- Apply Changes Permanently `firewall-cmd --runtime-to-permanent`
- Reload firewall to apply permanent changes `firewall-cmd –reload`

7. **Add custom service to firewall**

```
cp /usr/lib/firewalld/services/http.xml /etc/firewalld/services/
  mv /etc/firewalld/services/http.xml /etc/firewalld/services/abc.xml
  vim /etc/firewalld/services/abc.xml
  firewall-cmd --reload
  firewall-cmd --add-service=abc
  firewall-cmd --list-all
```

## Port Forwarding (honeypot)

Masquerading in firewalld is a form of network address translation (NAT) that allows devices on a private network to communicate with external networks using the IP address of the firewall's external interface. This is commonly used to provide internet access to a private network or to hide the internal network's structure from the outside world, enhancing security. Masquerading is especially useful in-home networks, small office environments, or for any setup where multiple devices need to share a single public IP address.

### Enabling Masquerading

To enable masquerading, you use the --add-masquerade option with firewall-cmd. If you're applying masquerading to allow internal devices to access the internet, you typically apply it to the external zone, often the public zone, but the exact configuration might vary depending on your network setup.

### Install and configure http service on both servers

- `yum install httpd* -y` install http on both server
- `vim /var/www/html/index.html` edit welcome page
- `systemctl start httpd`

Here's how to enable masquerading on Server1

- `sudo firewall-cmd --zone=public --add-masquerade --permanent`
- `sudo firewall-cmd --reload`

### Verifying Masquerading

To check if masquerading is enabled for a specific zone:

- `sudo firewall-cmd --zone=public --query-masquerade`

### Use Case: Masquerading with Port Forwarding

Masquerading can be combined with port forwarding to, for example, direct traffic from the outside world to a service hosted on an internal network. This could be part of setting up a honeypot, as in your earlier example. First, enable masquerading as shown above. Then, set up port forwarding.

For instance, to forward HTTP traffic (port 80) to a server at 192.168.244.55, you would use:

```
firewall-cmd --zone=public --add-forward-
port=port=80:proto=tcp:toport=80:toaddr=192.168.244.55 --permanent
firewall-cmd --zone=public --add-forward-
port=port=22:proto=tcp:toport=22:toaddr=192.168.244.55 --permanent
firewall-cmd –reload
```

---

---

**forward traffic from port 80 to 22**

```
firewall-cmd --zone=public --add-forward-
port=port=80:proto=tcp:toport=22:toaddr=192.168.244.55 –permanent
```

**List and remove port forward**

```
firewall-cmd --zone=public --list-forward-port
firewall-cmd --zone=public --remove-forward-
port=port=80:proto=tcp:toport=80:toaddr=192.168.244.55 --permanent
firewall-cmd --zone=public --remove-forward-
port=port=22:proto=tcp:toport=22:toaddr=192.168.244.55 --permanent
firewall-cmd --zone=public --remove-forward-
port=port=80:proto=tcp:toport=22:toaddr=192.168.244.55 --permanent
firewall-cmd –reload
```

## Managing Rich Rules with **firewalld**

Rich rules in firewalld provide a powerful and flexible way to define detailed firewall policies. These rules allow administrators to fine-tune traffic filtering with precision, going beyond basic port and service specifications. Here's an in-depth exploration of rich rules, including creating, listing, and deleting them, along with a specific example to block HTTP traffic from a designated network.

### What are Rich Rules?

Rich rules are advanced firewall rules that offer granular control over network traffic. They can specify conditions based on IP addresses, ports, protocols, and more, enabling actions like accept, reject, or drop. Rich rules can also incorporate logging and rate limiting.

### Adding a Rich Rule

To add a rich rule, use the firewall-cmd command with the --add-rich-rule option. The syntax requires specifying the zone and the rule in a single quoted string. For changes to persist after a reboot, add the --permanent flag.

**Example Syntax:**

```
firewall-cmd --zone=public --add-rich-rule='rule family="ipv4" source
address="192.168.244.0/24" reject' –permanent
firewall-cmd --reload
```

### Listing Rich Rules

- To view all rich rules for a given zone, use: `sudo firewall-cmd --zone=public --list-rich-rules`
- This command displays rich rules currently defined in the public zone.

### Deleting a Rich Rule

To remove a rich rule, use the --remove-rich-rule option, mirroring the --add-rich-rule syntax. For permanent removal, include --permanent and reload the firewall:

```
firewall-cmd --zone=public --remove-rich-rule='rule family="ipv4" source
address="192.168.1.0/24" port port="80" protocol="tcp" reject' --permanent
firewall-cmd –reload
```

---

---

**Example: Blocking HTTP Traffic from a Specific Network**

Consider a scenario where you need to block all HTTP (port 80) traffic from a particular network, say 192.168.244.0/24, to enhance your network security or manage traffic flow.

**Step 1:** Create the rich rule to block HTTP traffic:

```
firewall-cmd --zone=public --add-rich-rule='rule family="ipv4" source
address="192.168.244.0/24" port port="80" protocol="tcp" reject' –permanent
```

**Step 2:** Reload firewalld to apply the change: `sudo firewall-cmd –reload`

**Step 3:** Verify the rule has been added: `sudo firewall-cmd --zone=public --list-rich-rules`

---

# DHCP

**Install DHCP Service** Install the DHCP server package using `dnf`: `dnf install dhcp-server`
**Check DHCP Configuration Files** Verify the DHCP configuration files in `/etc/dhcp/`: `ls /etc/dhcp/`

**Check Last User Created** To view the last user created, which might be related to the DHCP service account, you can use: `tail -1 /etc/passwd`

**Configure DHCP Subnet Range and Server Configuration**

`vim /etc/dhcp/dhcpd.conf`
Alternatively, you can replace the existing configuration file with the example provided by the DHCP

`cp /usr/share/doc/dhcp-server/dhcpd.conf.example /etc/dhcp/dhcpd.conf`
Or, append the contents of the example configuration to the end of the current configuration file:

`cat /usr/share/doc/dhcp-server/dhcpd.conf.example >> /etc/dhcp/dhcpd.conf`
This step allows you to view the default configuration and possibly customize it according to your needs.

## Basic Server Configuration:

```
subnet 192.168.244.0 netmask 255.255.255.0 {
range 192.168.244.222 192.168.244.254;
option routers 192.168.244.2;
option subnet-mask 255.255.255.0;
option domain-name-servers 1.1.1.1, 8.8.8.8;
default-lease-time 600;
max-lease-time 7200;
}
```

**Remember**, after making changes to the DHCP configuration, you'll need to start (or restart) the DHCP service for your changes to take effect. You can do this with:

```
systemctl start dhcpd.service # To start the service
systemctl enable dhcpd.service ^start^enable # To enable the service at boot
systemctl restart dhcpd.service # To restart the service if it was already running
systemctl status dhcpd.service # verify it's running correctly
```

## Basic client commands

```
dhclient -r ens160 #release TCP/IP configuration
dhclient ens160 #renew TCP/IP configuration
cat /var/lib/dhclient/dhclient.leases
nmcli con show ens160
ip a s
cat /etc/resolv.conf
route -n
```

---

---

## DHCP Logs

To enable DHCP logs, include the `log-facility local5;` directive in your `/etc/dhcp/dhcpd.conf`. This tells the DHCP server to log its messages to the `local5` log facility.

**Configuring `rsyslog` to Handle DHCP Logs**

After setting `log-facility local5;` in the DHCP configuration, ensure that your `/etc/rsyslog.conf` file is configured to route `local5` logs to a specific file. If it's not already set up, you can add a line like this to `/etc/rsyslog.conf:`

```
Local5.*    /var/log/dhcpd.log
```

This configuration directs all messages from the local5 facility (where we've told DHCP to log its messages) to /var/log/dhcpd.log. After making changes to rsyslog.conf, remember to restart the rsyslog service:

```
systemctl restart rsyslog
```

## DHCP Reservation

```
host PC1 {
  hardware ethernet 00:1A:2B:3C:4D:5E; # The MAC address of the device
  fixed-address 192.168.1.201; # The IP address you want to assign to the device
}
```

## Securing DHCP Server: Denying Access to Unknown Clients

When you include the `deny unknown-clients;` statement in your DHCP configuration, it effectively blocks any device that isn't explicitly listed in the configuration from receiving an IP address. To allow a specific device access, you must define a host declaration for that device, specifying its hardware (MAC) address and the IP address you wish to assign to it.

```
host PC1 { hardware ethernet 00:1A:2B:3C:4D:5E;}
host PC2 { hardware ethernet 00:1A:2B:3C:4D:51;}
host PC3 { hardware ethernet 00:1A:2B:3C:4D:52;}
host PC4 { hardware ethernet 00:1A:2B:3C:4D:53;}
```

`cat /var/log/dhcp.log` #where you can see the denied requests.

## Securing DHCP Server: Blocking a Specific Computer from Accessing DHCP Services

```
host denied-pc { hardware ethernet 00:0c:29:78:10:ec; deny booting; }
```

## DHCP statistics

- **`cat /var/lib/dhcpd/dhcpd.leases`** #Displays active DHCP leases.
- **`cat /var/lib/dhcpd/dhcpd.leases~`** #Shows backup of the last DHCP leases state.
- **`echo "" > /var/lib/dhcpd/dhcpd.leases`** #Clears current DHCP leases data.
- **`echo "" > /var/lib/dhcpd/dhcpd.leases~`** #Empties backup DHCP leases file.

---

---

## DHCP statistics using script:

```
#!/bin/bash

LEASES_FILE="/var/lib/dhcpd/dhcpd.leases"
DHCPD_CONF="/etc/dhcp/dhcpd.conf" # Assuming the configuration file path
# Define the start and end of the DHCP range
start_ip="192.168.244.222"
end_ip="192.168.244.254"
# Function to convert IP to an integer
ip_to_int() {
    local IFS=.
    read ip1 ip2 ip3 ip4 <<< "$1"
    echo $((ip1 * 256 ** 3 + ip2 * 256 ** 2 + ip3 * 256 + ip4))
}

# Convert start and end IPs to integers
start_ip_int=$(ip_to_int $start_ip)
end_ip_int=$(ip_to_int $end_ip)

# Calculate total possible IPs in the range
total_ips=$((end_ip_int - start_ip_int + 1))

# Count the number of active leases
# This simplistic approach counts all lease declarations
active_leases=$(grep -c "lease" $LEASES_FILE)
# Calculate remaining IPs
remaining_ips=$((total_ips - active_leases))
echo "Lease Declarations: $active_leases"
echo "Total IPs in Range: $total_ips"
echo "Remaining IPs: $remaining_ips"
```

**Note:** It is recommended to run `systemctl restart dhcpd.service` before executing the script.

---

---

# DNS

- server name is **biglab**
- `vim /etc/nsswitch.conf` your Linux system to resolve hostnames by first checking local files (`/etc/hosts`), then DNS servers, and finally ensuring its own hostname resolves locally. This sequence prioritizes local definitions over network-based DNS queries for hostname resolution.
  - `dnf install bind bind-utils` (berkeley internet name domain) #install the BIND DNS server along with utilities for querying DNS servers (`dig, nslookup, etc`.).
  - `tail -3 /etc/passwd` #check dns service account (named)
  - `cp /etc/named.conf /etc/named.conf_bk` first backup the original the configuration file
  - `chgrp named /etc/named.conf.orig` change the group owner to be like the original one

```
[root@redlab ~]# ll /etc/named.conf
-rw-r-----. 1 root named 1705 Aug 19  2021 /etc/named.conf
[root@redlab ~]# ll /etc/named.conf.orig
-rw-r-----. 1 root root 1705 Mar 28 01:49 /etc/named.conf.orig
```

-

## named minimal configuration

`vim /etc/named.conf` #dns configuration file

A minimal configuration for a caching nameserver (which forwards requests to other DNS servers and caches the results) might look something like this:

```
options {
    listen-on port 53 { any; };
    listen-on-v6 port 53 { any; };
    directory "/var/named";
    dump-file "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    secroots-file "/var/named/data/named.secroots";
    allow-query { any; };

    /*
     - Forward requests to Google DNS or your preferred upstream DNS
     - Remove or comment out these lines if you're setting up an authoritative DNS server
    */
    forwarders {
        8.8.8.8;
        8.8.4.4;
    };

forward only; #also you can option forward first;

    recursion yes;
};

logging {
    channel default_debug {
        file "data/named.run";
```

---

```
        severity dynamic;
    };
};

zone "." IN {
    type hint;
    file "named.ca";
};

include "/etc/named.rfc1912.zones";
include "/etc/named.root.key";
```

## Start and Enable BIND Service

```
sudo systemctl start named
sudo systemctl enable named
sudo systemctl status named
sudo firewall-cmd --permanent --add-service=dns
sudo firewall-cmd –reload
```

## Change DNS client settings

```
nmcli con show
nmcli con mod ens160 ipv4.dns "192.168.244.111"
nmcli con mod ens160 ipv4.ignore-auto-dns yes
nmcli con down ens160; nmcli con up ens160
```

## Testing DNS Configuration

1. **Disable Recursion and Restrict Queries to localhost**
   - **Edit named.conf** on your DNS server:
     - Set recursion no; to disable recursion.
     - Add allow-query { localhost; }; to restrict DNS queries to the local machine.
   - **Restart BIND** to apply changes:
   - sudo systemctl restart named
2. **Test Configuration and DNS Resolution**
   - **Check Configuration Validity**:
   - `named-checkconf`
     This command verifies the syntax of your BIND configuration file. If there's no output, your configuration is likely correct.
   - **Test DNS Resolution with dig and nslookup**:
     - From your DNS server (acting as localhost):
     - dig @localhost google.com
     - nslookup google.com localhost
       Both commands should fail to resolve google.com due to recursion being disabled and queries being allowed only from localhost.
     - From another machine or client configured to use your DNS server:
       Try to resolve google.com with dig or nslookup. These should fail since the server only responds to queries from localhost.

---

3. **Enable Recursion for localhost Only**
   - **Update named.conf**
     - Change to recursion yes; to enable recursion.
     - Keep allow-query { localhost; }; to maintain query restriction to localhost.
   - **Restart BIND**.
4. **Repeat Tests**
   - **On the DNS Server (localhost)**:
     - Use dig or nslookup again:
     - dig @localhost google.com
     - nslookup google.com localhost
       Now, these should succeed in resolving google.com since recursion is enabled, but only for localhost.
   - **On Another Machine/Client**:
     - Attempts to resolve google.com will still fail because of the allow-query { localhost; }; setting, which prevents external queries.

**5. Test Without Forwarders:**

- With the forwarders disabled, your DNS server should now rely on the **root hints** file located at /var/named/named.ca to recursively resolve queries by contacting the root DNS servers and following the delegation chain down to the authoritative servers for the queried domain.

## DNS Caching

Try to run `time nslookup abdelwahed.me` twice you can the time decreased. The time difference you observe when running nslookup (or similar commands) consecutively is due to DNS caching.
`systemd-resolved`: This is becoming the standard DNS caching service on many Linux distributions. While there's no simple command to display the cache contents, you can query a specific domain to see if it's being resolved from the cache or network. Use `resolvectl` or `systemd-resolve` to query a domain:
`resolvectl query abdelwahed.me`
This doesn't show the entire cache but can test if a particular domain is cached. Also you use last command to check the result coming from network or from cache
While running `dig abdelwahed.me` shows TTL which the age of the cached record inside the DNS cache.
To clear the whole cache `sudo systemd-resolve --flush-caches`

## DNS Zones

### abdelwahed.me zone

To create another zone for `abdelwahed.me` hosted on the `biglab` DNS server and take initial zone data from `named.empty`, follow these steps:

**Step 1: Define the Zone in named.conf**

1. **Open your BIND Configuration File**:

sudo nano /etc/named.conf

2. **Add the Zone Definition:**

At the end of the file, add a zone block for abdelwahed.me:

```
zone "abdelwahed.me" IN {
    type master;
    file "/var/named/abdelwahed.me.db";
    allow-update { none; };
};
```

---

---

This tells BIND that you're setting up `abdelwahed.me` as a master zone, and its configuration will be stored in `/var/named/abdelwahed.me.db`. Adjust the file path as necessary for your system.

**Step 2: Create the Zone File**

1. **Create the Zone File**:

Based on the path provided in the `named.conf`, create the zone file:

sudo nano /var/named/abdelwahed.me.db

2. **Add DNS Records**:

In the zone file, add the necessary DNS records. Here's a simple example to get you started, including the `biglab` server:

```
$TTL 86400
@   IN  SOA biglab.abdelwahed.me. admin.abdelwahed.me. (
        2023100401 ; Serial
        3600       ; Refresh
        1800       ; Retry
        604800     ; Expire
        86400      ; Minimum TTL
    )
@           IN  NS      biglab.abdelwahed.me.
biglab      IN  A       192.168.244.111
@           IN  A       192.168.244.111
www         IN  A       192.168.244.111
blog        IN CNAME biglab.abdelwahed.me.
```

- **SOA Record**: Start of Authority, crucial for zone management. Replace `admin.abdelwahed.me.` with your actual administrative contact email, but replace the `@` with a `.`.
- **NS Record**: Declares `biglab.abdelwahed.me.` as the name server.

**A Records**: `biglab`, `@` (domain root), and `www` point to `192.168.244.111`.

**Step 3: Check Configuration and Restart BIND**

1. **Check Configuration for Errors**:

```
named-checkconf
named-checkzone abdelwahed.me /var/named/abdelwahed.me.db
```

2. **Restart BIND**:

```
sudo systemctl restart named
```

**Step 4: Test the DNS Setup**

After configuring the zone and restarting BIND, test your DNS setup:

```
dig @192.168.244.111 abdelwahed.me
dig @192.168.244.111 www.abdelwahed.me
```

These commands query your newly configured DNS server for the `abdelwahed.me` domain and its `www` subdomain. Ensure the responses reflect the configurations you've set.

## lab.com zone

To create another zone for `lab.com` hosted on the `biglab` DNS server and take initial zone data from `named.empty`, follow these steps:

**Step 1: Define the Zone in named.conf**

1. **Open Your BIND Configuration File**:

For a typical BIND installation:

```
sudo nano /etc/named.conf
```

2. **Add the Zone Definition for lab.com**:

Add a new zone block for lab.com to your BIND configuration file:

```
zone "lab.com" IN {
    type master;
    file "/var/named/lab.com.db";
    allow-update { none; };
};
```

---

---

This configuration specifies that `lab.com` is a master zone and its data will be stored in `/var/named/lab.com.db`. Ensure you adjust the file path based on your system's configuration and practices.

**Step 2: Create the Zone File for `lab.com`**

1. **Create the Zone File**:

Based on the path you specified in the `named.conf,` create the new zone file:

sudo nano /var/named/lab.com.db

2. **Add DNS Records to the Zone File**:

Here's a basic setup for the `lab.com` zone file:

```
$TTL 86400
@   IN  SOA ns1.lab.com. admin.lab.com. (
        2023100501 ; Serial
        3600       ; Refresh
        1800       ; Retry
        604800     ; Expire
        86400      ; Minimum TTL
    )
    IN  NS      ns1.lab.com.
ns1 IN  A       192.168.244.111
@   IN  A       192.168.244.112
www IN  A       192.168.244.112
```

- **SOA Record**: Start of Authority, setting `ns1.lab.com.` as the primary name server and `admin.lab.com.` as the contact email (with the `@` replaced by a dot).
- **NS Record**: Declares `ns1.lab.com.` as a name server for `lab.com`.

**A Records**: Defines the IP addresses for the root domain (`@`), `www` subdomain, and the name server (`ns1`).

**Step 3: Check Configuration and Restart BIND**

1. **Check the Configuration for Errors**:

```
named-checkconf
named-checkzone lab.com /var/named/lab.com.db
```

2. **Restart BIND**:

```
sudo systemctl restart named
```

**Step 4: Test the New Zone**

After configuring and restarting BIND, test the new DNS setup for `lab.com`:

```
dig @192.168.244.111 lab.com
dig @192.168.244.111 www.lab.com
```

These commands query your DNS server for the `lab.com` domain and its `www` subdomain, verifying the configurations you've set.

`sudo rndc reload abdelwahed.me` #to update dns zone without restart named service

## authoritative or non-authoritative response

To determine whether a DNS response is authoritative or non-authoritative, you can use DNS lookup tools like dig or nslookup. An authoritative answer comes directly from a DNS server that has the authoritative data for the domain, meaning it's the ultimate source of information for that specific domain. A non-authoritative answer, on the other hand, comes from a server that doesn't hold the authoritative data but has obtained the information from another source, often caching it from previous queries.

---

---

**Using dig Command**

The dig command is very informative and explicitly indicates whether an answer is authoritative.

1. **Run dig**:
   Execute dig with the domain you're interested in. For example:
   dig abdelwahed.me
2. **Inspect the flags in the output**:
   In the response section, look for the flags part. If the answer is authoritative, you'll see the aa flag (which stands for "Authoritative Answer"). If the aa flag is missing, the answer is non-authoritative.
   Example of authoritative response flags:
   ```
   ;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3
   ```
   If the aa flag is absent, it indicates a non-authoritative answer.

## Conditional forwarders

This is useful in scenarios where you manage a local DNS server but want queries for certain domains to be resolved by an external DNS server.

**Configure Conditional Forwarding**

1. **Edit the BIND Configuration File**: Open /etc/named.conf or the appropriate include file if you're using a split configuration model.
   ```
   sudo vi /etc/named.conf
   ```
2. **Add a Forward Zone for Conditional Forwarding**: You need to define a forward zone within the named.conf file for each domain you want to conditionally forward. Here is an example configuration that forwards queries for abdelwahed.me to a specific DNS server:

```
zone "yahoo.com" IN {
    type forward;
    forward only; // or `forward first;` depending on your preference
    forwarders {
        1.1.1.1; // IP address of the DNS server to forward queries to
   };
};
```

   - type forward; declares that this zone is a forwarding zone.
   - forward only; means that queries will be forwarded and not resolved locally (you can also use forward first; to try local resolution if the forwarder fails).
   - forwarders { ... }; lists the DNS servers to which queries for the domain should be forwarded.
3. **Repeat for Additional Domains**: If you have more domains for which you want to set up conditional forwarding, add additional zone sections following the same format.

**3. Check the Configuration for Errors**

Before applying your changes, it's a good idea to check the BIND configuration for syntax errors:

```
sudo named-checkconf
```

If this command returns without output, the configuration syntax is correct.

**4. Restart BIND**

To apply your changes, restart the BIND service:

```
sudo systemctl restart named
```

**5. Verify Conditional Forwarding**

Test the conditional forwarding setup by querying a domain you've set up for forwarding:

```
dig @localhost subdomain.yahoo.com
dig +noall +answer blog.yahoo.com
```

The @localhost part directs dig to query your local BIND server. Replace subdomain.yahoo.com with an actual domain under the forwarded domain to test.

---

---

## Configuring Forwarding and Security

**1. Restrict Zone Transfers**

```
zone "abdelwahed.me" IN {
    type master;
    file "abdelwahed.me.db";
    allow-transfer { 192.0.2.1; }; // IP of authorized secondary DNS server
};
```

**2. Disable or Limit Recursion**

To disable recursion:

```
options {
    recursion no; };
```

To limit recursion to known clients:

```
options {
    recursion yes;
    allow-recursion { 192.0.2.0/24; }; // Subnet allowed to perform recursion
}; # you can use option localnets to allow quries from different subnets.
```

**3. Enable Response Rate Limiting (RRL)**

```
options {
    rate-limit {
        responses-per-second 5;
        window 5;
    }; };
```

**4. Enable DNSSEC**

```
zone "abdelwahed.me" IN {
    type master;
    file "abdelwahed.me.db";
    key-directory "keys/abdelwahed.me";
    auto-dnssec maintain;
    inline-signing yes;
};
```

**5. Configure Logging**

```
logging {
    channel default_file {
        file "logs/named.log" versions 3 size 5m;
        severity info;
        print-time yes;
    };
    category default { default_file; };
};
```

**6. Firewall Configuration**

# Allow incoming DNS queries

```
iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p tcp --dport 53 -j ACCEPT
```

# Block unwanted traffic

```
iptables -A INPUT -j DROP
```

**7. Run DNS Services with Least Privileges**

Ensure BIND runs as a non-root user, typically named: # Typically configured by default in BIND installations

**8. Backup and Recovery Plan**

Regularly backup your BIND configuration and zone files:

```
cp /etc/named.conf /backup/location/named.conf.backup
cp /var/named/* /backup/location/
```

---

---

## DNS zone types

DNS zone types define how a particular zone is managed and its operational role within the DNS infrastructure. Here are the primary DNS zone types with examples:

1. **Primary Zone (Master Zone)**

A primary zone is the authoritative source for information about a particular domain. It is where the domain's DNS records are created, stored, and maintained manually or through dynamic updates.

- **Example Configuration in BIND**:

```
zone "abdelwahed.me" IN {
    type master;
    file "/etc/bind/db.abdelwahed.me";
};
```

This defines abdelwahed.me as a primary zone, with its data stored in the specified file.

2. **Secondary Zone (Slave Zone)**

A secondary zone is a read-only copy of the primary zone. It gets its data through zone transfers from the primary server. Secondary zones provide redundancy and load distribution.

- **Example Configuration in BIND**:

```
zone "abdelwahed.me" IN {
    type slave;
    masters { 192.0.2.1; }; // IP address of the primary server
    file "db.abdelwahed.me.bak";
};
```

This setup creates a secondary zone for `abdelwahed.me`, which obtains its data from the primary server located at `192.0.2.1`.

3. **Stub Zone**

A stub zone contains only a small portion of the authoritative zone, specifically the NS records of the master zone, a copy of the SOA record, and possibly glue records. It's used to track authoritative servers for a domain, reducing the need for recursive queries.

- **Example Configuration in BIND**:

```
zone "abdelwahed.me" IN {
    type stub;
    masters { 192.0.2.1; };
    file "db.abdelwahed.me.stub";
};
```

This configuration creates a stub zone that tracks the authoritative NS records for `abdelwahed.me` from the primary server at `192.0.2.1`.

4. **Forward Zone**

A forward zone redirects queries for a specific domain to another DNS server. It's useful when you want queries to be handled by a specific server, possibly for policy or architecture reasons.

- **Example Configuration in BIND**:

```
zone "abdelwahed.me" IN {
    type forward;
    forwarders { 192.0.2.2; 192.0.2.3; };
};
```

Queries for abdelwahed.me are forwarded to the DNS servers 192.0.2.2 and 192.0.2.3.

---

## Reverse DNS zones

To create reverse DNS zones for the IP addresses listed in your zone data, you first need to identify the reverse zone name for the IP subnet. For the IP addresses `192.168.244.111` and `192.168.244.202` in the subnet `192.168.244.0/24`, the reverse zone would be for the `244.168.192.in-addr.arpa` domain.

Here's how you can define reverse DNS zones for `abdelwahed.me` and `lab.com`, assuming both are using IP addresses from the same subnet.

**1. Define the Reverse Zone in named.conf**

Open your BIND DNS server's main configuration file, `named.conf`, and add a reverse zone entry. This example assumes you're managing the entire `192.168.244.0/24` subnet:

```
zone "244.168.192.in-addr.arpa" IN {
    type master;
    file "/var/named/244.168.192.in-addr.arpa.db";
    allow-update { none; };
};
```

**2. Create the Reverse Zone File**

Create a new zone file at `/var/named/244.168.192.in-addr.arpa.db` as specified in your reverse zone definition. Here's how the contents might look:

```
$TTL 86400
@   IN  SOA biglab.abdelwahed.me. admin.abdelwahed.me. (
        2023100401 ; Serial
        3600       ; Refresh
        1800       ; Retry
        604800     ; Expire
        86400      ; Minimum TTL
    )
    IN  NS  biglab.abdelwahed.me.

111 IN  PTR biglab.abdelwahed.me.
111 IN  PTR biglab.lab.com.
111 IN  PTR www.abdelwahed.me.
111 IN  PTR test.abdelwahed.me.
202 IN  PTR client.abdelwahed.me.
202 IN  PTR portal.abdelwahed.me.
```

Note that each PTR record maps an IP address back to a hostname. The left side of each PTR record is the last octet of the IP address in the subnet, since this reverse zone covers the `192.168.244.0/24` subnet.

**3. Check Configuration and Reload BIND**

After creating or modifying zone files, it's always a good idea to check your configuration for errors:

```
sudo named-checkconf
sudo named-checkzone "244.168.192.in-addr.arpa" /var/named/244.168.192.in-addr.arpa.db
```

If everything is configured correctly, reload the BIND service to apply the changes: `systemctl reload named`

---

# Web Server

## Installing the Apache HTTPD Server

- `dnf search httpd`
- `dnf install httpd httpd-manual`
- `systemctl status httpd.service`
- `systemctl start httpd.service`
- `systemctl enable httpd.service`
- `pgrep httpd`
- `apachectl configtest`
- `dnf install -y epel-release`
- `dnf install -y w3m`
- `w3m http://127.0.0.1`
- `firewall-cmd --permanent --add-service=http`
- `firewall-cmd --permanent --add-service=https`
- `firewall-cmd –reload`
- `netstat -ltn`
- `firewall-cmd --list-services`

## Working with the Apache Configuration (Basic Configuration)

**1. Editing the Server Name:**

- Open the Apache configuration file with `vim`: `vim /etc/httpd/conf/httpd.conf`
- Search for the `ServerName` directive. If it's not present, add it by including your domain or IP. For example: `ServerName biglab.abdelwahed.me`

**2. Configuring Listen Ports:**

- You can change the default Listen port (`80`) to another one according to your requirements. To listen on multiple ports or specific IP addresses, configure as follows:
- `Listen 80`
- `Listen 443`
  For a specific IP and port:
- `Listen 192.168.244.111:80`
  This ensures that all incoming connections must come through the specified IP address.

**3. Changing the Home Page:**

- To set a custom home page, you can redirect the output to your index.html file: `echo "<h1>ABDELWAHED FIRST SITE</h1>" > /var/www/html/index.html`
- If you want to use a different home page, adjust the `DirectoryIndex` directive in the configuration file to the new file name.

**4. Applying the Changes:**

- Restart the Apache service to apply your changes: `systemctl restart httpd`

**5. Adding Custom HTML Content:**

- You can customize the content of your website by directly editing or adding files in the web directory `(/var/www/html/).` For example, you could add an HTML file with the following content to set a personalized welcome message and design for your site:

Snippet of the HTML template you might use:

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome to Red Hat World</title>
```

---

---

```
    <style>
        body {
            background-color: orange;
            font-family: Arial, sans-serif;
        }
        h1, h4 {
            text-align: center;
            color: #333; /* Dark grey text color for better readability */
        }
    </style>
</head>
<body>

<h1>Welcome to Red Hat World</h1>

<h4>Nice to Join Us</h4>
</body>
</html>
```

This snippet shows a basic HTML structure with a styled body, header, and title that you could adapt for your site. Always ensure that any custom HTML or configuration changes are thoroughly tested to avoid disrupting your web service

## Creating Virtual Hosts on Apache Web Server

**Introduction to Name-based Virtual Hosting:**

Name-based, or virtual hosting, allows you to host multiple domains on a single Apache web server. This setup is efficient when you have various domain names pointing to your server's IP address, facilitated through DNS server configurations or local /etc/hosts file entries. Each virtual host directs users to a specific directory containing the website's data.

**Initial Setup:**

- **DNS Configuration**: Ensure all websites' DNS records point to the server's IP address or manually add them to the /etc/hosts file for local resolution.

**Configuration for a Portal Website:**

1. **Create the Configuration File**:
    - Navigate to the `conf.d` directory, which serves as the root for server configurations.
    - Create a new configuration file for your portal website: `vim /etc/httpd/conf.d/portal.conf`

```
<VirtualHost *:80>
 ServerName "portal.abdelwahed.me"
 DocumentRoot "/var/www/portal"
</VirtualHost>
```

2. **Set Up the Document Root Directory**:
    - Create a directory to store the website's files: `mkdir -p /var/www/portal`
3. **Create the Website's Index File**:
    - Create and edit the `index.html` file to add your website's content: `vim /var/www/portal/index.html`
    - Here's a basic template for the website's homepage:

```
<!DOCTYPE html>
<html>
<body style="background-color:lightgrey;">
<h1 style="text-align:center;">WELCOME TO REDHAT Portal</h1>
<h4 style="text-align:center;">NICE TO JOIN US</h4>
</body>
</html>
```

---

---

**Testing the Configuration**:

- After setting up your virtual host, it's crucial to test the Apache configuration for any errors: `apachectl configtest`
- This command will verify the syntax of your configuration files. If you receive a "Syntax OK" message, your configuration has no immediate errors.

**Applying Changes**:

- Finally, to apply your changes, restart the Apache service: `systemctl restart httpd`
- This ensures that your Apache server recognizes the new virtual host and starts serving the portal website according to the configurations you've set.

## Creating a Second Virtual Host for the Main Website

**Step 1: Set Up Virtual Host Configuration**

- Navigate to the conf.d directory, which houses the web server's configuration files.
- Create the main website's configuration file:
  `vim /etc/httpd/conf.d/main.conf`
- Insert the following configuration to define the virtual host for the main website:

```
<VirtualHost *:80>
ServerName "main.abdelwahed.me"
DocumentRoot "/var/www/main"
</VirtualHost>
```

**Step 2: Create Document Root Directory**

- Prepare a directory to store the main website's files: `mkdir -p /var/www/main`

**Step 3: Add Website Content**

- Create an index.html file for the main website with initial content: `vim /var/www/main/index.html`
- You can start with a basic HTML structure like this:

```
<!DOCTYPE html>
<html>
<body style="background-color:lightgrey;">
<h1 style="text-align:center;">WELCOME TO REDHAT Main</h1>
<h4 style="text-align:center;">NICE TO JOIN US</h4>
</body>
</html>
```

**Final Configuration Check**

- Ensure your configuration is correct by running: `apachectl configtest`

Additional Configuration for Security and Status Page

**Security Measure: Prevent .html File Override**

- To enhance security and prevent users from overriding existing files, specify AllowOverride none within a <Directory> directive in your main website's configuration file (/etc/httpd/conf.d/main.conf).

## Adding a Status Directory

```
vim /etc/httpd/conf.d/main.conf:
<Location "/server-status">
    SetHandler server-status
    Require host localhost
    # Or use Require ip 192.168.1.0/24 for allowing access from a specific network
</Location>
```

After making these changes, validate the configuration and restart Apache:

```
apachectl configtest
systemctl restart httpd
```

**Testing the Configuration**

- For the main site: `w3m main.abdelwahed.me`
- For the status page: `w3m main.abdelwahed.me/status`

---

---

## Configuring Log Files for Individual Hosts

- `/etc/httpd/logs`, default access and error logs

To configure custom log files for an individual virtual host, such as a branch virtual host, follow these instructions:

1. **Edit the Virtual Host Configuration:** Open the virtual host configuration file for editing. For example, if we're configuring the branch virtual host, you might edit a `branch.conf` file within `/etc/httpd/conf.d/`. However, for demonstration, we'll use `main.conf:`
2. `vi /etc/httpd/conf.d/main.conf`
   Add the following lines to specify custom access and error log paths:
   ```
   ErrorLog "logs/main_error"
   CustomLog "logs/main_access" common
   ```
   Here, `"common"` specifies the log format.
3. **Test Configuration:** Check the Apache configuration for any syntax errors: `apachectl configtest`
4. **Restart Apache:** Apply the changes by restarting Apache: `systemctl restart httpd`
5. **Verify Log Files:** After restarting Apache, check `/etc/httpd/logs`. You should find the `main_access` and `main_error` files reflecting access and error logs for the branch virtual host.
   To test these changes, access the branch web from any computer, then run:
   ```
   cat /etc/httpd/logs/main_access
   ```
   You will find all access logs related to this branch site.

## Restricting Access by Username

To restrict access to a specific directory by username and password, follow these steps:

1. **Edit the Configuration:** Open or create a configuration file for the branch virtual host in /etc/httpd/conf.d/:
2. `vim /etc/httpd/conf.d/main.conf`
   Add the following lines to secure a directory with Basic authentication:
   ```
   <Directory "/var/www/main">
       Require valid-user
       AuthType Basic
       AuthName "Private Access"
       AuthBasicProvider file
       AuthUserFile "/etc/httpd/conf.d/main"
   </Directory>
   ```
3. **Create a Password File:** Move to the /etc/httpd/conf.d/ directory and create a file to store usernames and passwords:
   ```
   cd /etc/httpd/conf.d/
   htpasswd -c main ahmed
   ```
   To add additional users, omit -c after the first time:
   ```
   htpasswd main user1
   ```
4. **Test Configuration and Restart Apache:** Use apachectl configtest for syntax checking and systemctl restart httpd to apply the changes.
5. **Verify Users and Passwords:** Use cat main in the /etc/httpd/conf.d/ directory to view the users and hashed passwords

---

# Red Hat System Administration III RH254

---

## IP-Based Restrictions for Server Status

1. **Configure IP Restrictions**:
   - Edit `/etc/httpd/conf.d/branch.conf` to restrict access to the server status:
   - vi /etc/httpd/conf.d/branch.conf
   - Add the following to restrict access to /status to specific IPs:

```
<Location /status>
 SetHandler server-status
 Require ip 127.0.0.1
</Location>
```

2. **Validate and Restart Apache**:
   - Use `apachectl configtest` to check configuration, then `systemctl restart httpd` to apply changes.
3. **Test IP Restrictions**:
   - Attempt to access `http://your_server_ip/status` from both allowed and disallowed IPs to verify the restrictions. Check error logs for unauthorized access attempts:

```
tail -f /etc/httpd/logs/branch_error
```

## Change http port to 8080

To change the Apache HTTP Server to listen on port 8080 instead of the default port 80 on a Red Hat Enterprise Linux (RHEL) system, you'll need to modify the Apache configuration files and adjust any associated firewall settings to allow traffic on the new port. Here's how to do it:

1. Modify Apache Configuration

1. **Open the Apache Configuration File:** The main Apache configuration file is typically named `httpd.conf` and located in `/etc/httpd/conf/.` There might also be a `ports.conf` file in the same directory or under `/etc/httpd/conf.d/` if your setup is configured to use it. If `ports.conf` exists, it's a better practice to make your changes there.

```
sudo vi /etc/httpd/conf/httpd.conf
```

2. **Find the Listen Directive:** Look for the line that specifies the `Listen` directive. It will likely have the default setting of `80`. Change this line to listen on port 8080:

```
Listen 8080
```

3. **Update Virtual Hosts (if applicable):** If you have virtual hosts configured in `/etc/httpd/conf.d/` or within the `httpd.conf` file, update the `<VirtualHost>` directives to reflect the new port number. For example, change:

```
<VirtualHost *:80>
To:
<VirtualHost *:8080>
```

4. **Save and Close the File:** After making these changes, save the file and exit the text editor.

2. Adjust SELinux Settings (If SELinux is Enforcing)

```
sudo semanage port -a -t http_port_t -p tcp 8080
```

3. Update Firewall Settings

```
sudo firewall-cmd --permanent --zone=public --add-port=8080/tcp
sudo firewall-cmd --reload
```

4. Restart Apache

```
sudo systemctl restart httpd
```

5. Verify the Change

You can verify that Apache is listening on the new port by using the curl command or a web browser to access your server using port 8080. With `curl`, it would look like this:

curl http://localhost:8080

Or simply open `http://localhost:8080` or `http://your-server-ip:8080` in your web browser, replacing `localhost or your-server-ip` with your actual server IP address or domain name.

---

---

## Enabling SSL on Your Website

Using SSL (Secure Sockets Layer) is crucial, especially if your site involves user authentication or handles sensitive data. SSL not only encrypts data between the client and the server, ensuring privacy and security, but also authenticates the server to confirm users are connecting to the legitimate site. Here's how you can enable SSL on your site:

1. Install mod_ssl

Start by installing the mod_ssl module, which adds SSL support to Apache:

```
dnf install -y mod_ssl
```

After installation, the SSL configuration file will be located in /etc/httpd/conf.d.

2. Prepare the SSL Configuration File

Rename the default SSL configuration file to use it as a template:

```
mv /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/ssl.template
```

3. Generate SSL Certificate and Key

Navigate to the Apache configuration directory:

```
cd /etc/httpd/conf.d
```

Generate a new SSL certificate and private key with the following command. This will create them without a passphrase for convenience:

```
openssl req -new -nodes -x509 -keyout main.key -out main.crt -nodes
```

4. Secure the Certificate and Key

```
chmod 400 main.key main.crt
```

5. Configure Your Site for SSL

```
vim main.conf
```

Add the following configuration, adjusting ServerName, DocumentRoot, and paths as necessary:

```
<VirtualHost *:443>
    ServerName "main.abdelwahed.me"
    DocumentRoot "/var/www/main"
    SSLEngine on
    SSLCertificateKeyFile "conf.d/main.key"
    SSLCertificateFile "conf.d/main.crt"
    <Directory "/var/www/main">
    </Directory>
</VirtualHost>
```

6. Test and Restart Apache

```
apachectl configtest
systemctl restart httpd
```

7. Verify Port 443 is Listening

```
netstat -ltn
```

8. Test Your Site

```
w3m https://main.abdelwahed.me
```

9. Update Firewall and SELinux Settings

Ensure port 443 is allowed through your firewall. For SELinux, either configure it to allow HTTPS traffic or temporarily disable it for testing (not recommended for production environments).

To allow port 443 through the firewall:

```
firewall-cmd --permanent --zone=public --add-port=443/tcp
firewall-cmd --reload
```

To adjust SELinux for testing:

```
setenforce 0
```

(Remember to set it back to enforcing mode with setenforce 1 after testing.)

Final Steps

Optionally, you may want to redirect HTTP traffic to HTTPS by modifying the virtual host for port 80 in /etc/httpd/conf/httpd.conf to include a redirect to your HTTPS site.

---

---

# SAMBA

## Basic Setup: Simple File Sharing

**Install Samba**

- **Update your RHEL System**: `sudo dnf update`
- **Install Samba**: `dnf install samba samba-client`

**Configure Samba**

- **Edit Configuration**: Modify /etc/samba/smb.conf by adding:

```
[PublicShare]
path = /share
browseable = yes
read only = no
guest ok = yes
```

Start and Enable Samba Services

```
systemctl start smb nmb
systemctl enable smb nmb
```

Firewall Configuration

Allow Samba through the firewall:

```
firewall-cmd --permanent --add-service=samba
firewall-cmd --reload
```

- **Check Samba Configuration**: `testparm`
- **Access SMB Share**: `smbclient //localhost/share -U aabdelwahed`

## Secure File Sharing

```
Create a Samba User
useradd smbuser1
smbpasswd -a smbuser1
smbpasswd -a root
```

**Configure Secure Share**

Add to /etc/samba/smb.conf:

```
[SecureShare]
path = /secureshare
valid users = smbuser1
browseable = yes
writable = yes
```

Restart Samba Services `systemctl restart smb nmb`

Access from Windows and Linux

Authentication will be required. Use myuser and the password you set.

## Differentiating Permissions

Adjust /etc/samba/smb.conf for different access levels:

```
[SecureShare]
   path = /secureshare2
   browseable = yes
   writable = yes
   valid users = user1 user2 user3
   write list = user1 user3
   read list = user2
```

- **user1 and user3** have read-write access.
- **user2** has read-only access.

---

---

Using Filesystem Permissions or ACLs
Setting Basic Permissions

```
chown user1:user1 /share
chmod 755 /share
```

## Using ACLs for Detailed Permissions

Ensure filesystem has ACL support, then:

```
setfacl -m u:user1:rw /share
setfacl -m u:user2:r /share
```

Apply default ACLs for new files/directories.

**Note**Samba adheres to the most restrictive permissions set by its configuration or the Linux filesystem/ACLs.

## Mounting the Shared Folder on Linux

1. **Install cifs-utils** `dnf install cifs-utils`

```
mount -t cifs //192.168.244.123/share /mnt/samba_share -o
username=smbuser1,password=12345
```

2. **Create a Credentials File** Store your credentials securely in `vi /etc/samba/credentials,`

```
username=myusername
password=mypassword
```

3. set: `chmod 600 /etc/samba/user.cred`
4. `mount -t cifs //192.168.244.123/share /mnt/samba_share -o cred=/etc/cifs-creds`
5. **Create a Mount Point** `sudo mkdir -p /mnt/myshare`
6. **Mount the Share** Use `mount.cifs` to mount the share at `/mnt/myshare.`
7. **Verify the Mount** Ensure the share is correctly mounted with `df -h` or `mount | grep cifs`

## Making the Mount Permanent

vi /etc/systemd/system/mnt-samba_share.mount

```
[Unit]
Description=My Web Server
After=network.target
[Service]
Type=simple
ExecStart=/usr/bin/mywebserver
Restart=on-failure
RestartSec=10s
User=www-data
Environment="PORT=8080" "CONFIG=/etc/mywebserver/config.json"
[Install]

WantedBy=multi-user.target
```

To auto-mount at boot, add the share to `/etc/fstab`:

```
//server/sharename /mnt/myshare cifs
credentials=/etc/samba/user.cred,iocharset=utf8,vers=3.0,_netdev 0 0
```

**Important**: Verify your `/etc/fstab` entries to prevent boot issues. Backup `/etc/fstab` before editing.

---

---

# MariaDB

## Step 1: Install MariaDB

1. **Update Your System**: Ensure your system packages are up to date. `sudo yum update -y`
2. **Install MariaDB Server**: Install the MariaDB package using `yum`. `sudo yum install mariadb-server -y`
3. **Start MariaDB Service**: Once the installation is complete, start the MariaDB service. `systemctl start mariadb`
4. **Enable MariaDB on Boot**: To ensure MariaDB starts on boot. `sudo systemctl enable mariadb`
5. **Open MariaDB Port:** By default, MariaDB listens on port 3306. You need to open this port in the firewall to allow incoming connections.

```
sudo firewall-cmd --zone=public --add-port=3306/tcp --permanent
sudo firewall-cmd –reload
sudo firewall-cmd --zone=public --list-all
```

## Step 2: Secure MariaDB Installation

Run the `mysql_secure_installation` script to set the root password, disable remote root logins, and remove test databases.

```
sudo mysql_secure_installation
```

Follow the prompts to secure your installation. For a new installation, you might not have a root password, so press `Enter` when asked for one, and then set a new root password when prompted.

## Step 3: Log into MariaDB

You can log into the MariaDB shell as the root user by running: `mysql -u root -p`
Enter the root password you set during the secure installation process.

## Step 4: Example Operations in MariaDB

### Changing the Root Password

If you need to change the root password for your MariaDB server, you can do so by logging into the MariaDB shell and issuing the `ALTER USER` command.

```
mysql -u root -p
```

Once logged in:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'pass123';
FLUSH PRIVILEGES;
```

### Creating a New Database User

Creating a new user allows you to manage access to your databases effectively.

```
CREATE USER 'ahmed'@'localhost' IDENTIFIED BY '333';
```

### Granting Permissions to a User

After creating a user, you need to grant them specific permissions on databases or tables.

```
GRANT ALL PRIVILEGES ON db1.* TO 'ahmed'@'localhost';
FLUSH PRIVILEGES;
```

### Creating a Database

```
CREATE DATABASE db1;
```

### Listing All Databases `SHOW DATABASES;`

### Selecting a Database

To start working with a specific database:

```
USE db1;
```

---

---

### Creating a Table

To create a new table within a selected database:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(50),
    joined DATE
      );
```

### Inserting Data into a Table

To insert data into a table:

```
INSERT INTO users (username, email, joined) VALUES
('jane_doe', 'jane@example.com', '2022-02-01'),
('alex_reed', 'alex@example.com', '2022-03-01'),
('sam_curran', 'sam@example.com', '2022-04-01'),
('lily_bush', 'lily@example.com', '2022-05-01'),
('mark_zoe', 'mark@example.com', '2022-06-01'),
('sophia_laren', 'sophia@example.com', '2022-07-01'),
('daniel_craig', 'daniel@example.com', '2022-08-01'),
('lucas_brown', 'lucas@example.com', '2022-09-01'),
('emma_watson', 'emma@example.com', '2022-10-01'),
('oliver_twist', 'oliver@example.com', '2022-11-01');
```

Retrieve Data:

```
SELECT * FROM users;
```

Update Data:

```
UPDATE users SET email = 'newjohn@example.com' WHERE username = 'jane_doe';
```

Delete Data:

```
DELETE FROM users WHERE username = 'jane_doe';
```

Drop the Database via SQL Commands:

```
DROP DATABASE IF EXISTS db1;
```

Exit MariaDB shell:

```
EXIT;
```

Backing Up a Database

```
mysqldump -u root -p db1 > db1_backup.sql
```

Restoring a Database from a Backup

```
mysql -u root -p db1 < db1_backup.sql
```

Note: Backup and restore is not an SQL statement but a command-line utility used for backing up MySQL or MariaDB databases. You cannot run it directly within the SQL command line or MariaDB shell. Instead, it must be executed from your server's command-line interface (CLI).

While it's technically possible to delete a database by directly removing its data directory (e.g., `/var/lib/mysql/db1/`) using a command like `rm -rf /var/lib/mysql/db1/`, and then recreate the database with the same name to restore from a backup, this method is highly discouraged due to the risk of data corruption, potential data loss, compromised database integrity, permissions issues, and possible service interruption. These actions bypass the database management system's safeguards, risking permanent data inconsistency, corruption, and other unintended consequences.

### Viewing Running Queries

To see what queries are currently running on your MariaDB server:

```
SHOW FULL PROCESSLIST;
```

### Dropping a Database

```
DROP DATABASE db1;
```

### Dropping a User

```
DROP USER 'dbuser'@'localhost';
```

---