



Red Hat Enterprise Linux 9

Configuring firewalls and packet filters

Managing the firewalld service, the nftables framework, and XDP packet filtering features

Red Hat Enterprise Linux 9 Configuring firewalls and packet filters

Managing the firewalld service, the nftables framework, and XDP packet filtering features

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Packet filters, such as firewalls, use rules to control incoming, outgoing, and forwarded network traffic. In Red Hat Enterprise Linux (RHEL), you can use the `firewalld` service and the `nftables` framework to filter network traffic and build performance-critical firewalls. You can also use the Express Data Path (XDP) feature of the kernel to process or drop network packets at the network interface at a very high rate.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. USING AND CONFIGURING FIREWALLD	6
1.1. WHEN TO USE FIREWALLD, OR NFTABLES	6
1.2. FIREWALL ZONES	6
1.3. FIREWALL POLICIES	8
1.4. FIREWALL RULES	9
1.5. FIREWALL DIRECT RULES	9
1.6. PREDEFINED FIREWALLD SERVICES	10
1.7. WORKING WITH FIREWALLD ZONES	10
1.7.1. Customizing firewall settings for a specific zone to enhance security	10
1.7.2. Changing the default zone	12
1.7.3. Assigning a network interface to a zone	12
1.7.4. Adding a source	13
1.7.5. Removing a source	13
1.7.6. Assigning a zone to a connection using nmcli	14
1.7.7. Creating a new zone	14
1.7.8. Enabling zones by using the web console	14
1.7.9. Disabling zones by using the web console	16
1.7.10. Using zone targets to set default behavior for incoming traffic	17
1.7.11. Configuring dynamic updates for allowlisting with IP sets	18
1.8. CONTROLLING NETWORK TRAFFIC USING FIREWALLD	19
1.8.1. Controlling traffic with predefined services using the CLI	19
1.8.2. Enabling services on the firewall by using the web console	21
1.8.3. Configuring custom ports by using the web console	22
1.9. FILTERING FORWARDED TRAFFIC BETWEEN ZONES	24
1.9.1. The relationship between policy objects and zones	24
1.9.2. Using priorities to sort policies	25
1.9.3. Using policy objects to filter traffic between locally hosted containers and a network physically connected to the host	25
1.9.4. Setting the default target of policy objects	26
1.10. CONFIGURING NAT USING FIREWALLD	26
1.10.1. Network address translation types	27
1.10.2. Configuring IP address masquerading	27
1.10.3. Using DNAT to forward incoming HTTP traffic	28
1.10.4. Redirecting traffic from a non-standard port to make the web service accessible on a standard port	29
1.11. PRIORITIZING RICH RULES	30
1.11.1. How the priority parameter organizes rules into different chains	31
1.11.2. Setting the priority of a rich rule	31
1.12. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE	32
1.12.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT	32
1.12.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network	32
1.13. CONFIGURING FIREWALLD BY USING RHEL SYSTEM ROLES	33
1.13.1. Resetting the firewall settings by using the firewall RHEL system role	34
1.13.2. Forwarding incoming traffic in firewalld from one local port to a different local port by using the firewall RHEL system role	35
1.13.3. Configuring a firewall DMZ zone by using the firewall RHEL system role	36
CHAPTER 2. GETTING STARTED WITH NFTABLES	39
2.1. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES	39
2.1.1. Basics of nftables tables	39

2.1.2. Basics of nftables chains	40
Chain types	40
Chain priorities	41
Chain policies	41
2.1.3. Basics of nftables rules	41
2.1.4. Managing tables, chains, and rules using nft commands	42
2.2. MIGRATING FROM IPTABLES TO NFTABLES	44
2.2.1. When to use firewalld, or nftables	45
2.2.2. Concepts in the nftables framework	45
2.2.3. Concepts in the deprecated iptables framework	46
2.2.4. Converting iptables and ip6tables rule sets to nftables	47
2.2.5. Converting single iptables and ip6tables rules to nftables	48
2.2.6. Comparison of common iptables and nftables commands	49
2.3. CONFIGURING NAT USING NFTABLES	49
2.3.1. NAT types	50
2.3.2. Configuring masquerading using nftables	50
2.3.3. Configuring source NAT using nftables	51
2.3.4. Configuring destination NAT using nftables	51
2.3.5. Configuring a redirect using nftables	52
2.4. WRITING AND EXECUTING NFTABLES SCRIPTS	53
2.4.1. Supported nftables script formats	53
2.4.2. Running nftables scripts	54
2.4.3. Using comments in nftables scripts	55
2.4.4. Using variables in nftables script	55
2.4.5. Including files in nftables scripts	56
2.4.6. Automatically loading nftables rules when the system boots	56
2.5. USING SETS IN NFTABLES COMMANDS	57
2.5.1. Using anonymous sets in nftables	57
2.5.2. Using named sets in nftables	58
2.5.3. Using dynamic sets to add entries from the packet path	59
2.5.4. Additional resources	60
2.6. USING VERDICT MAPS IN NFTABLES COMMANDS	60
2.6.1. Using anonymous maps in nftables	60
2.6.2. Using named maps in nftables	62
2.6.3. Additional resources	63
2.7. EXAMPLE: PROTECTING A LAN AND DMZ USING AN NFTABLES SCRIPT	63
2.7.1. Network conditions	64
2.7.2. Security requirements to the firewall script	64
2.7.3. Configuring logging of dropped packets to a file	65
2.7.4. Writing and activating the nftables script	66
2.8. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS	69
2.8.1. Limiting the number of connections by using nftables	69
2.8.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute	70
2.9. DEBUGGING NFTABLES RULES	70
2.9.1. Creating a rule with a counter	70
2.9.2. Adding a counter to an existing rule	71
2.9.3. Monitoring packets that match an existing rule	72
2.10. BACKING UP AND RESTORING THE NFTABLES RULE SET	72
2.10.1. Backing up the nftables rule set to a file	73
2.10.2. Restoring the nftables rule set from a file	73
2.11. ADDITIONAL RESOURCES	73

ATTACKS	74
3.1. DROPPING NETWORK PACKETS THAT MATCH AN XDP-FILTER RULE	74
3.2. DROPPING ALL NETWORK PACKETS EXCEPT THE ONES THAT MATCH AN XDP-FILTER RULE	75

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the [Jira](#) website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. USING AND CONFIGURING FIREWALLD

A *firewall* is a way to protect machines from any unwanted traffic from outside. It enables users to control incoming network traffic on host machines by defining a set of *firewall rules*. These rules are used to sort the incoming traffic and either block it or allow through.

firewalld is a firewall service daemon that provides a dynamic customizable host-based firewall with a D-Bus interface. Being dynamic, it enables creating, changing, and deleting the rules without the necessity to restart the firewall daemon each time the rules are changed.

firewalld uses the concepts of zones and services, that simplify the traffic management. Zones are predefined sets of rules. Network interfaces and sources can be assigned to a zone. The traffic allowed depends on the network your computer is connected to and the security level this network is assigned. Firewall services are predefined rules that cover all necessary settings to allow incoming traffic for a specific service and they apply within a zone.

Services use one or more ports or addresses for network communication. Firewalls filter communication based on ports. To allow network traffic for a service, its ports must be open. **firewalld** blocks all traffic on ports that are not explicitly set as open. Some zones, such as trusted, allow all traffic by default.

Note that **firewalld** with **nftables** backend does not support passing custom **nftables** rules to **firewalld**, using the **--direct** option.

1.1. WHEN TO USE FIREWALLD, OR NFTABLES

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance-critical firewalls, such as for a whole network.



IMPORTANT

To prevent the different firewall-related services (**firewalld**, or **nftables**) from influencing each other, run only one of them on a RHEL host, and disable the other services.

1.2. FIREWALL ZONES

You can use the **firewalld** utility to separate networks into different zones according to the level of trust that you have with the interfaces and traffic within that network. A connection can only be part of one zone, but you can use that zone for many network connections.

firewalld follows strict principles in regards to zones:

1. Traffic ingresses only one zone.
2. Traffic egresses only one zone.
3. A zone defines a level of trust.
4. Intrazone traffic (within the same zone) is allowed by default.
5. Interzone traffic (from zone to zone) is denied by default.

Principles 4 and 5 are a consequence of principle 3.

Principle 4 is configurable through the zone option **--remove-forward**. Principle 5 is configurable by adding new policies.

NetworkManager notifies **firewalld** of the zone of an interface. You can assign zones to interfaces with the following utilities:

- **NetworkManager**
- **firewall-config** utility
- **firewall-cmd** utility
- The RHEL web console

The RHEL web console, **firewall-config**, and **firewall-cmd** can only edit the appropriate **NetworkManager** configuration files. If you change the zone of the interface using the web console, **firewall-cmd**, or **firewall-config**, the request is forwarded to **NetworkManager** and is not handled by **firewalld**.

The **/usr/lib/firewalld/zones/** directory stores the predefined zones, and you can instantly apply them to any available network interface. These files are copied to the **/etc/firewalld/zones/** directory only after they are modified. The default settings of the predefined zones are as follows:

block

- Suitable for: Any incoming network connections are rejected with an icmp-host-prohibited message for **IPv4** and icmp6-adm-prohibited for **IPv6**.
- Accepts: Only network connections initiated from within the system.

dmz

- Suitable for: Computers in your DMZ that are publicly-accessible with limited access to your internal network.
- Accepts: Only selected incoming connections.

drop

Suitable for: Any incoming network packets are dropped without any notification.

- Accepts: Only outgoing network connections.

external

- Suitable for: External networks with masquerading enabled, especially for routers. Situations when you do not trust the other computers on the network.
- Accepts: Only selected incoming connections.

home

- Suitable for: Home environment where you mostly trust the other computers on the network.
- Accepts: Only selected incoming connections.

internal

- Suitable for: Internal networks where you mostly trust the other computers on the network.
- Accepts: Only selected incoming connections.

public

- Suitable for: Public areas where you do not trust other computers on the network.
- Accepts: Only selected incoming connections.

trusted

- Accepts: All network connections.

work

Suitable for: Work environment where you mostly trust the other computers on the network.

- Accepts: Only selected incoming connections.

One of these zones is set as the *default* zone. When interface connections are added to **NetworkManager**, they are assigned to the default zone. On installation, the default zone in **firewalld** is the **public** zone. You can change the default zone.

**NOTE**

Make network zone names self-explanatory to help users understand them quickly.

To avoid any security problems, review the default zone configuration and disable any unnecessary services according to your needs and risk assessments.

Additional resources

- **firewalld.zone(5)** man page on your system

1.3. FIREWALL POLICIES

The firewall policies specify the desired security state of your network. They outline rules and actions to take for different types of traffic. Typically, the policies contain rules for the following types of traffic:

- Incoming traffic
- Outgoing traffic
- Forward traffic
- Specific services and applications
- Network address translations (NAT)

Firewall policies use the concept of firewall zones. Each zone is associated with a specific set of firewall rules that determine the traffic allowed. Policies apply firewall rules in a stateful, unidirectional manner. This means you only consider one direction of the traffic. The traffic return path is implicitly allowed due

to stateful filtering of **firewalld**.

Policies are associated with an ingress zone and an egress zone. The ingress zone is where the traffic originated (received). The egress zone is where the traffic leaves (sent).

The firewall rules defined in a policy can reference the firewall zones to apply consistent configurations across multiple network interfaces.

1.4. FIREWALL RULES

You can use the firewall rules to implement specific configurations for allowing or blocking network traffic. As a result, you can control the flow of network traffic to protect your system from security threats.

Firewall rules typically define certain criteria based on various attributes. The attributes can be as:

- Source IP addresses
- Destination IP addresses
- Transfer Protocols (TCP, UDP, ...)
- Ports
- Network interfaces

The **firewalld** utility organizes the firewall rules into zones (such as **public**, **internal**, and others) and policies. Each zone has its own set of rules that determine the level of traffic freedom for network interfaces associated with a particular zone.

1.5. FIREWALL DIRECT RULES

The **firewalld** service provides multiple ways with which to configure rules, including:

- regular rules
- direct rules

One difference between these is how each method interacts with the underlying backend (**iptables** or **nftables**).

The direct rules are advanced, low-level rules that allow direct interaction with **iptables**. They bypass the structured zone-based management of **firewalld** to give you more control. You manually define the direct rules with the **firewall-cmd** command by using the raw **iptables** syntax. For example, **firewall-cmd --direct --add-rule ipv4 filter INPUT 0 -s 198.51.100.1 -j DROP**. This command adds an **iptables** rule to drop traffic from the 198.51.100.1 source IP address.

However, using the direct rules also has its drawbacks. Especially when **nftables** is your primary firewall backend. For example:

- The direct rules are harder to maintain and can conflict with **nftables** based **firewalld** configurations.
- The direct rules do not support advanced features that you can find in **nftables** such as raw expressions and stateful objects.

- Direct rules are not future-proof. The **iptables** component is deprecated and will eventually be removed from RHEL.

For the previous reasons, you might consider replacing **firewalld** direct rules with **nftables**. Review the knowledgebase solution [How to replace firewalld direct rules with nftables?](#) to see more details.

1.6. PREDEFINED FIREWALLD SERVICES

The predefined **firewalld** services provide a built-in abstraction layer over the low-level firewall rules. It is achieved by mapping commonly used network services, such as SSH or HTTP to their corresponding ports and protocols. Instead of manually specifying these each time, you can refer to a named predefined service. This makes firewall management simpler, less error-prone, and more intuitive.

- To see available predefined services:

```
# firewall-cmd --get-services
```

```
RH-Satellite-6 RH-Satellite-6-capsule afp amanda-client amanda-k5-client amqp amqps
apcupsd audit ausweisapp2 bacula bacula-client bareos-director bareos-filedaemon bareos-
storage bb bgp bitcoin bitcoin-rpc bitcoin-testnet bitcoin-testnet-rpc bittorrent-lsd ceph ceph-
exporter ceph-mon cfengine checkmk-agent cockpit collectd condor-collector cratedb ctdb
dds...
```

- To further inspect a particular predefined service:

```
# sudo firewall-cmd --info-service=RH-Satellite-6
```

```
RH-Satellite-6
ports: 5000/tcp 5646-5647/tcp 5671/tcp 8000/tcp 8080/tcp 9090/tcp
protocols:
source-ports:
modules:
destination:
includes: foreman
helpers:
```

The example output shows that the **RH-Satellite-6** predefined service listens on ports 5000/tcp 5646-5647/tcp 5671/tcp 8000/tcp 8080/tcp 9090/tcp. Additionally, **RH-Satellite-6** inherits rules from another predefined service. In this case **foreman**.

Each predefined service is stored as an XML file with the same name in the **/usr/lib/firewalld/services/** directory.

Additional resources

- **firewall-cmd(1)**, **firewalld(1)** manual pages

1.7. WORKING WITH FIREWALLD ZONES

Zones represent a concept to manage incoming traffic more transparently. The zones are connected to networking interfaces or assigned a range of source addresses. You manage firewall rules for each zone independently, which enables you to define complex firewall settings and apply them to the traffic.

1.7.1. Customizing firewall settings for a specific zone to enhance security

You can strengthen your network security by modifying the firewall settings and associating a specific network interface or connection with a particular firewall zone. By defining granular rules and restrictions for a zone, you can control inbound and outbound traffic based on your intended security levels.

For example, you can achieve the following benefits:

- Protection of sensitive data
- Prevention of unauthorized access
- Mitigation of potential network threats

Prerequisites

- The **firewalld** service is running.

Procedure

1. List the available firewall zones:

```
# firewall-cmd --get-zones
```

The **firewall-cmd --get-zones** command displays all zones that are available on the system, but it does not show any details for particular zones. To see more detailed information for all zones, use the **firewall-cmd --list-all-zones** command.

2. Choose the zone you want to use for this configuration.
3. Modify firewall settings for the chosen zone. For example, to allow the **SSH** service and remove the **ftp** service:

```
# firewall-cmd --add-service=ssh --zone=<your_chosen_zone>
# firewall-cmd --remove-service=ftp --zone=<same_chosen_zone>
```

4. Assign a network interface to the firewall zone:

- a. List the available network interfaces:

```
# firewall-cmd --get-active-zones
```

Activity of a zone is determined by the presence of network interfaces or source address ranges that match its configuration. The default zone is active for unclassified traffic but is not always active if no traffic matches its rules.

- b. Assign a network interface to the chosen zone:

```
# firewall-cmd --zone=<your_chosen_zone> --change-interface=<interface_name> -
-permanent
```

Assigning a network interface to a zone is more suitable for applying consistent firewall settings to all traffic on a particular interface (physical or virtual).

The **firewall-cmd** command, when used with the **--permanent** option, often involves updating NetworkManager connection profiles to make changes to the firewall configuration permanent. This integration between **firewalld** and NetworkManager ensures

consistent network and firewall settings.

Verification

1. Display the updated settings for your chosen zone:

```
# firewall-cmd --zone=<your_chosen_zone> --list-all
```

The command output displays all zone settings including the assigned services, network interface, and network connections (sources).

1.7.2. Changing the default zone

System administrators assign a zone to a networking interface in its configuration files. If an interface is not assigned to a specific zone, it is assigned to the default zone. After each restart of the **firewalld** service, **firewalld** loads the settings for the default zone and makes it active. Note that settings for all other zones are preserved and ready to be used.

Typically, zones are assigned to interfaces by NetworkManager according to the **connection.zone** setting in NetworkManager connection profiles. Also, after a reboot NetworkManager manages assignments for "activating" those zones.

Prerequisites

- The **firewalld** service is running.

Procedure

To set up the default zone:

1. Display the current default zone:

```
# firewall-cmd --get-default-zone
```

2. Set the new default zone:

```
# firewall-cmd --set-default-zone <zone_name>
```



NOTE

Following this procedure, the setting is a permanent setting, even without the **--permanent** option.

1.7.3. Assigning a network interface to a zone

It is possible to define different sets of rules for different zones and then change the settings quickly by changing the zone for the interface that is being used. With multiple interfaces, a specific zone can be set for each of them to distinguish traffic that is coming through them.

Procedure

To assign the zone to a specific interface:

1. List the active zones and the interfaces assigned to them:

■


```
# firewall-cmd --get-active-zones
```

2. Assign the interface to a different zone:

```
# firewall-cmd --zone=zone_name --change-interface=interface_name --permanent
```

1.7.4. Adding a source

To route incoming traffic into a specific zone, add the source to that zone. The source can be an IP address or an IP mask in the classless inter-domain routing (CIDR) notation.



NOTE

In case you add multiple zones with an overlapping network range, they are ordered alphanumerically by zone name and only the first one is considered.

- To set the source in the current zone:

```
# firewall-cmd --add-source=<source>
```

- To set the source IP address for a specific zone:

```
# firewall-cmd --zone=zone-name --add-source=<source>
```

The following procedure allows all incoming traffic from *192.168.2.15* in the **trusted** zone:

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
```

2. Add the source IP to the trusted zone in the permanent mode:

```
# firewall-cmd --zone=trusted --add-source=192.168.2.15
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.7.5. Removing a source

When you remove a source from a zone, the traffic which originates from the source is no longer directed through the rules specified for that source. Instead, the traffic falls back to the rules and settings of the zone associated with the interface from which it originates, or goes to the default zone.

Procedure

1. List allowed sources for the required zone:

```
# firewall-cmd --zone=zone-name --list-sources
```

2. Remove the source from the zone permanently:

```
# firewall-cmd --zone=zone-name --remove-source=<source>
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

1.7.6. Assigning a zone to a connection using nmcli

You can add a **firewalld** zone to a **NetworkManager** connection using the **nmcli** utility.

Procedure

1. Assign the zone to the **NetworkManager** connection profile:

```
# nmcli connection modify profile connection.zone zone_name
```

2. Activate the connection:

```
# nmcli connection up profile
```

1.7.7. Creating a new zone

To use custom zones, create a new zone and use it just like a predefined zone. New zones require the **--permanent** option, otherwise the command does not work.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create a new zone:

```
# firewall-cmd --permanent --new-zone=zone-name
```

2. Make the new zone usable:

```
# firewall-cmd --reload
```

The command applies recent changes to the firewall configuration without interrupting network services that are already running.

Verification

- Check if the new zone is added to your permanent settings:

```
# firewall-cmd --get-zones --permanent
```

1.7.8. Enabling zones by using the web console

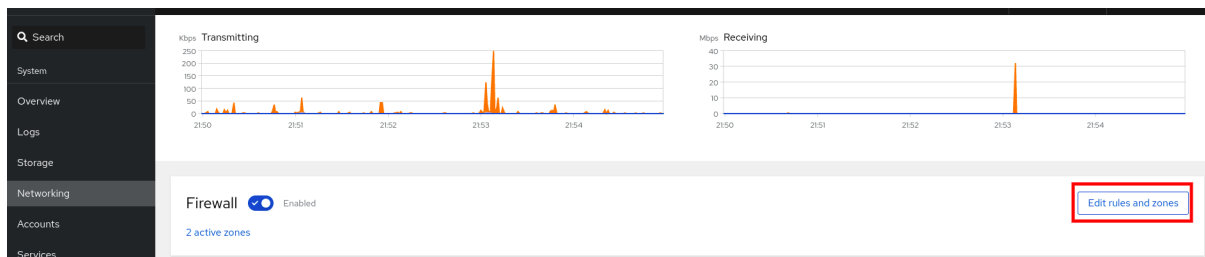
You can apply predefined and existing firewall zones on a particular interface or a range of IP addresses through the RHEL web console.

Prerequisites

- You have installed the RHEL 9 web console.
- You have enabled the cockpit service.
- Your user account is allowed to log in to the web console.
For instructions, see [Installing and enabling the web console](#).

Procedure

1. Log in to the RHEL 9 web console.
For details, see [Logging in to the web console](#).
2. Click **Networking**.
3. Click on the **Edit rules and zones** button.



If you do not see the **Edit rules and zones** button, log in to the web console with the administrator privileges.

4. In the **Firewall** section, click **Add new zone**.
5. In the **Add zone** dialog box, select a zone from the **Trust level** options.
The web console displays all zones predefined in the **firewalld** service.
6. In the **Interfaces** part, select an interface or interfaces on which the selected zone is applied.
7. In the **Allowed Addresses** part, you can select whether the zone is applied on:
 - the whole subnet
 - or a range of IP addresses in the following format:
 - 192.168.1.0
 - 192.168.1.0/24
 - 192.168.1.0/24, 192.168.1.0
8. Click on the **Add zone** button.

Add zone

Trust level

Sorted from least to most trusted

Custom zones

☐ Public

☐ External

☐ Dmz

☐ Work

☒ Home

☐ Internal

☐ FedoraServer

Description

For use in home areas. You mostly trust the other computers on networks to not harm your computer. Only selected incoming connections are accepted.

Included services

ssh, mdns, samba-client, dhcpv6-client

The cockpit service is automatically included

Interfaces

☐ enp0s20f0u4u1u2

☒ enp0s31f6

☐ p2p-dev-wlp61s0

☐ tap0

☐ tun0

Allowed addresses

☒ Entire subnet

☐ Range

Add zone

Cancel

Verification

- Check the configuration in the **Firewall** section:

Networking > Firewall

Firewall

☒ Enabled

Incoming requests are blocked by default. Outgoing requests are not blocked.

Add new zone

Home Zone

Interface enp0s31f6

Allowed addresses Entire subnet

Add services

Service

TCP

UDP

> ssh

22

> mdns

5353

> samba-client

137,138

> dhcpv6-client

546

> cockpit

9090

1.7.9. Disabling zones by using the web console

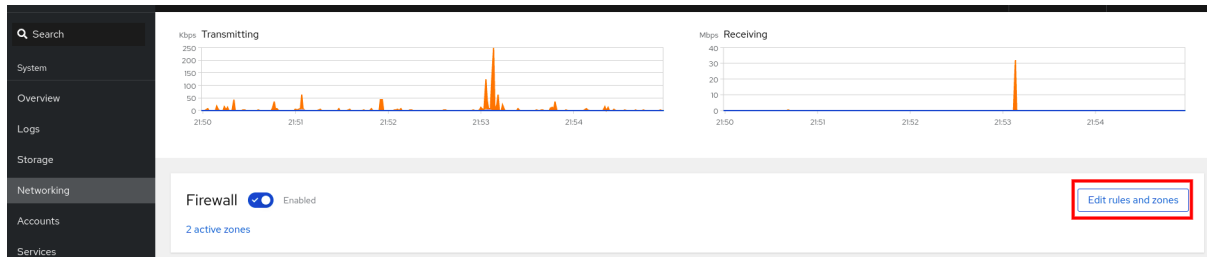
You can disable a firewall zone in your firewall configuration by using the web console.

Prerequisites

- You have installed the RHEL 9 web console.
- You have enabled the cockpit service.
- Your user account is allowed to log in to the web console.
For instructions, see [Installing and enabling the web console](#) .

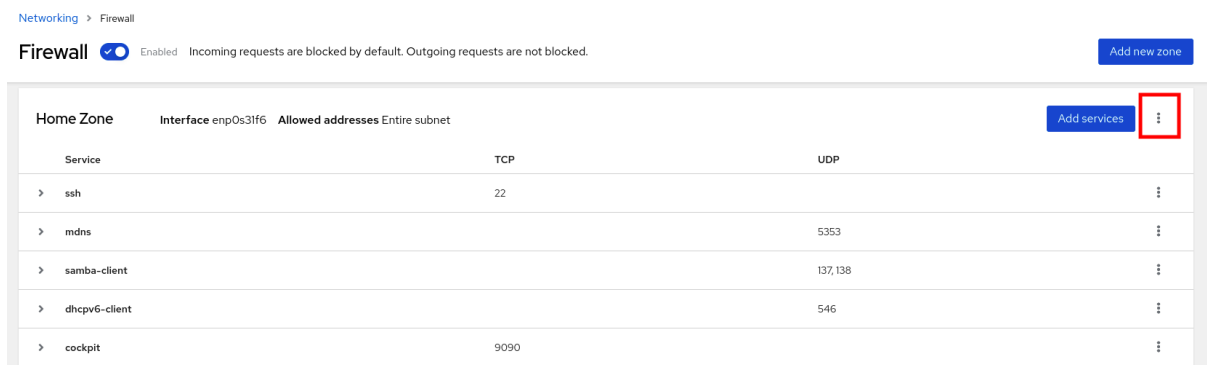
Procedure

1. Log in to the RHEL 9 web console.
For details, see [Logging in to the web console](#).
2. Click **Networking**.
3. Click on the **Edit rules and zones** button.



If you do not see the **Edit rules and zones** button, log in to the web console with the administrator privileges.

4. Click on the **Options** icon at the zone you want to remove.



5. Click **Delete**.

The zone is now disabled and the interface does not include opened services and ports which were configured in the zone.

1.7.10. Using zone targets to set default behavior for incoming traffic

For every zone, you can set a default behavior that handles incoming traffic that is not further specified. Such behavior is defined by setting the target of the zone. There are four options:

- **ACCEPT**: Accepts all incoming packets except those disallowed by specific rules.
- **REJECT**: Rejects all incoming packets except those allowed by specific rules. When **firewalld** rejects packets, the source machine is informed about the rejection.
- **DROP**: Drops all incoming packets except those allowed by specific rules. When **firewalld** drops packets, the source machine is not informed about the packet drop.
- **default**: Similar behavior as for **REJECT**, but with special meanings in certain scenarios.

Prerequisites

- The **firewalld** service is running.

Procedure

To set a target for a zone:

1. List the information for the specific zone to see the default target:

```
# firewall-cmd --zone=zone-name --list-all
```

2. Set a new target in the zone:

```
# firewall-cmd --permanent --zone=zone-name --set-target=  
<default|ACCEPT|REJECT|DROP>
```

Additional resources

- **firewall-cmd(1)** man page on your system

1.7.11. Configuring dynamic updates for allowlisting with IP sets

You can make near real-time updates to flexibly allow specific IP addresses or ranges in the IP sets even in unpredictable conditions. These updates can be triggered by various events, such as detection of security threats or changes in the network behavior. Typically, such a solution leverages automation to reduce manual effort and improve security by responding quickly to the situation.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create an IP set with a meaningful name:

```
# firewall-cmd --permanent --new-ipset=allowlist --type=hash:ip
```

The new IP set called **allowlist** contains IP addresses that you want your firewall to allow.

2. Add a dynamic update to the IP set:

```
# firewall-cmd --permanent --ipset=allowlist --add-entry=198.51.100.10
```

This configuration updates the **allowlist** IP set with a newly added IP address that is allowed to pass network traffic by your firewall.

3. Create a firewall rule that references the previously created IP set:

```
# firewall-cmd --permanent --zone=public --add-source=ipset:allowlist
```

Without this rule, the IP set would not have any impact on network traffic. The default firewall policy would prevail.

4. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

1. List all IP sets:

```
# firewall-cmd --get-ipsets
allowlist
```

2. List the active rules:

```
# firewall-cmd --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp0s1
  sources: ipset:allowlist
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  ...
```

The **sources** section of the command-line output provides insights to what origins of traffic (hostnames, interfaces, IP sets, subnets, and others) are permitted or denied access to a particular firewall zone. In this case, the IP addresses contained in the **allowlist** IP set are allowed to pass traffic through the firewall for the **public** zone.

3. Explore the contents of your IP set:

```
# cat /etc/firewalld/ipsets/allowlist.xml
<?xml version="1.0" encoding="utf-8"?>
<ipset type="hash:ip">
  <entry>198.51.100.10</entry>
</ipset>
```

Next steps

- Use a script or a security utility to fetch your threat intelligence feeds and update **allowlist** accordingly in an automated fashion.

Additional resources

- **firewall-cmd(1)** manual page

1.8. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

The **firewalld** package installs a large number of predefined service files and you can add more or customize them. You can then use these service definitions to open or close ports for services without knowing the protocol and port numbers they use.

1.8.1. Controlling traffic with predefined services using the CLI

The most straightforward method to control traffic is to add a predefined service to **firewalld**. This opens all necessary ports and modifies other settings according to the *service definition file*.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Check that the service in **firewalld** is not already allowed:

```
# firewall-cmd --list-services
ssh dhcpv6-client
```

The command lists the services that are enabled in the default zone.

2. List all predefined services in **firewalld**:

```
# firewall-cmd --get-services
RH-Satellite-6 amanda-client amanda-k5-client bacula bacula-client bitcoin bitcoin-rpc
bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine condor-collector ctdb dhcp dhcpv6
dhcpv6-client dns docker-registry ...
```

The command displays a list of available services for the default zone.

3. Add the service to the list of services that **firewalld** allows:

```
# firewall-cmd --add-service=<service_name>
```

The command adds the specified service to the default zone.

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The command applies these runtime changes to the permanent configuration of the firewall. By default, it applies these changes to the configuration of the default zone.

Verification

1. List all permanent firewall rules:

```
# firewall-cmd --list-all --permanent
public
target: default
icmp-block-inversion: no
interfaces:
sources:
services: cockpit dhcpv6-client ssh
ports:
protocols:
forward: no
masquerade: no
forward-ports:
source-ports:
icmp-blocks:
rich rules:
```

The command displays complete configuration with the permanent firewall rules of the default firewall zone (**public**).

2. Check the validity of the permanent configuration of the **firewalld** service.


```
# firewall-cmd --check-config
success
```

If the permanent configuration is invalid, the command returns an error with further details:

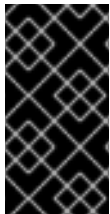
```
# firewall-cmd --check-config
Error: INVALID_PROTOCOL: 'public.xml': 'tcp' not from {'tcp'|'udp'|'sctp'|'dccp'}
```

You can also manually inspect the permanent configuration files to verify the settings. The main configuration file is `/etc/firewalld/firewalld.conf`. The zone-specific configuration files are in the `/etc/firewalld/zones/` directory and the policies are in the `/etc/firewalld/policies/` directory.

1.8.2. Enabling services on the firewall by using the web console

By default, services are added to the default firewall zone. If you use more firewall zones on more network interfaces, you must select a zone first and then add the service with port.

The RHEL 9 web console displays predefined **firewalld** services and you can add them to active firewall zones.



IMPORTANT

The RHEL 9 web console configures the **firewalld** service.

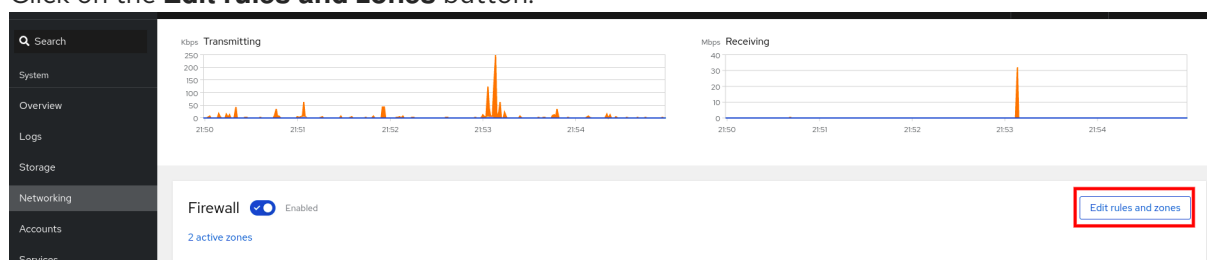
The web console does not allow generic **firewalld** rules which are not listed in the web console.

Prerequisites

- You have installed the RHEL 9 web console.
- You have enabled the cockpit service.
- Your user account is allowed to log in to the web console.
For instructions, see [Installing and enabling the web console](#).

Procedure

1. Log in to the RHEL 9 web console.
For details, see [Logging in to the web console](#).
2. Click **Networking**.
3. Click on the **Edit rules and zones** button.



If you do not see the **Edit rules and zones** button, log in to the web console with the administrator privileges.

- In the **Firewall** section, select a zone for which you want to add the service and click **Add Services**.

Networking > Firewall

Firewall ☒ Enabled Incoming requests are blocked by default. Outgoing requests are not blocked.

Add new zone

Home Zone		Interface enp0s31f6	Allowed addresses Entire subnet	Add services	
Service	TCP	UDP			
> ssh	22				
> mdns		5353			
> samba-client		137,138			
> dhcpv6-client		546			
> cockpit	9090				

- In the **Add Services** dialog box, find the service you want to enable on the firewall.
- Enable services according to your scenario:

Add services to home zone

×

☒ Services ☐ Custom ports

Filter services

freeIPA

- ☒ freeipa-4
TCP: 80, 443, 88, 464, 389, 636 UDP: 88, 464
- ☒ freeipa-ldap
TCP: 80, 443, 88, 464, 389 UDP: 88, 464, 123
- ☒ freeipa-ldaps
TCP: 80, 443, 88, 464, 636 UDP: 88, 464, 123
- ☐ freeipa-replication

Add services

Cancel

- Click **Add Services**.

At this point, the RHEL 9 web console displays the service in the zone's list of **Services**.

1.8.3. Configuring custom ports by using the web console

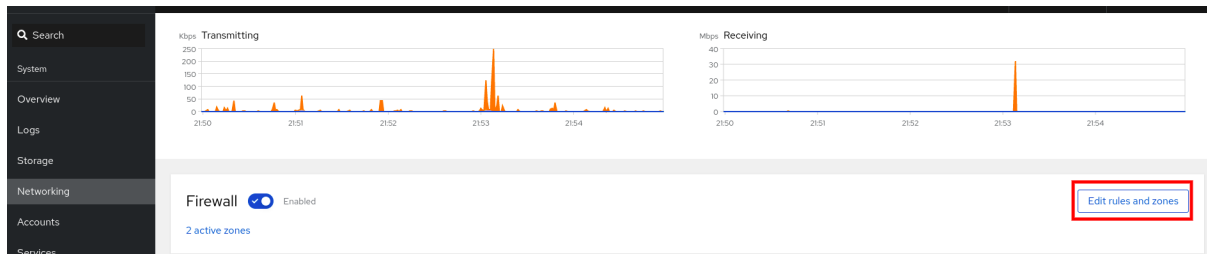
You can add configure custom ports for services through the RHEL web console.

Prerequisites

- You have installed the RHEL 9 web console.
- You have enabled the cockpit service.
- Your user account is allowed to log in to the web console.
For instructions, see [Installing and enabling the web console](#).
- The **firewalld** service is running.

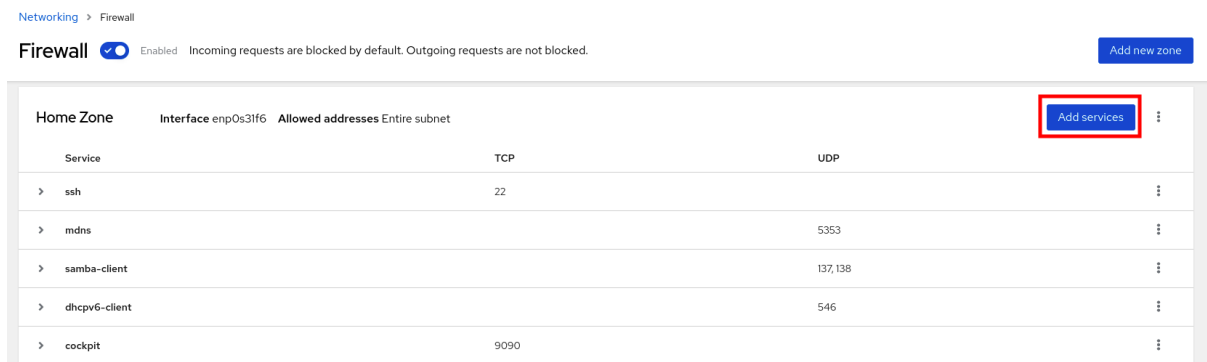
Procedure

1. Log in to the RHEL 9 web console.
For details, see [Logging in to the web console](#).
2. Click **Networking**.
3. Click on the **Edit rules and zones** button.



If you do not see the **Edit rules and zones** button, log in to the web console with the administrative privileges.

4. In the **Firewall** section, select a zone for which you want to configure a custom port and click **Add Services**.



5. In the **Add services** dialog box, click on the **Custom Ports** radio button.
6. In the TCP and UDP fields, add ports according to examples. You can add ports in the following formats:
 - Port numbers such as 22
 - Range of port numbers such as 5900-5910
 - Aliases such as nfs, rsync



NOTE

You can add multiple values into each field. Values must be separated with the comma and without the space, for example: 8080,8081,http

7. After adding the port number in the **TCP** field, the **UDP** field, or both, verify the service name in the **Name** field.
The **Name** field displays the name of the service for which is this port reserved. You can rewrite the name if you are sure that this port is free to use and no server needs to communicate on this port.
8. In the **Name** field, add a name for the service including defined ports.

9. Click on the **Add Ports** button.

Add ports to home zone

×

☐ Services
 ☒ Custom ports

TCP

Example: 22,ssh,8080,5900-5910

Comma-separated ports, ranges, and services are accepted

UDP

Example: 88,2019,nfs,rsync

Comma-separated ports, ranges, and services are accepted

ID

If left empty, ID will be generated based on associated port services and port numbers

Description

⚠ Adding custom ports will reload firewalld. A reload will result in the loss of any runtime-only configuration!

Add ports
Cancel

To verify the settings, go to the **Firewall** page and find the service in the list of zone's **Services**.

Networking > Firewall

Firewall ☒ Enabled Incoming requests are blocked by default. Outgoing requests are not blocked. Add new zone

Home Zone		Interface enp0s31f6	Allowed addresses Entire subnet	
Service	TCP	UDP		
> ssh	22			⋮
> mdns		5353		⋮
> samba-client		137, 138		⋮
> dhcpv6-client		546		⋮
> cockpit	9090			⋮

Add services ⋮

1.9. FILTERING FORWARDED TRAFFIC BETWEEN ZONES

firewalld enables you to control the flow of network data between different **firewalld** zones. By defining rules and policies, you can manage how traffic is allowed or blocked when it moves between these zones.

The policy objects feature provides forward and output filtering in **firewalld**. You can use **firewalld** to filter traffic between different zones to allow access to locally hosted VMs to connect the host.

1.9.1. The relationship between policy objects and zones

Policy objects allow the user to attach **firewalld**'s primitives such as services, ports, and rich rules to the policy. You can apply the policy objects to traffic that passes between zones in a stateful and unidirectional manner.

```
# firewall-cmd --permanent --new-policy myOutputPolicy
```

```
# firewall-cmd --permanent --policy myOutputPolicy --add-ingress-zone HOST
```

```
# firewall-cmd --permanent --policy myOutputPolicy --add-egress-zone ANY
```

HOST and **ANY** are the symbolic zones used in the ingress and egress zone lists.

- The **HOST** symbolic zone allows policies for the traffic originating from or has a destination to the host running firewalld.
- The **ANY** symbolic zone applies policy to all the current and future zones. **ANY** symbolic zone acts as a wildcard for all zones.

1.9.2. Using priorities to sort policies

Multiple policies can apply to the same set of traffic, therefore, priorities should be used to create an order of precedence for the policies that may be applied.

To set a priority to sort the policies:

```
# firewall-cmd --permanent --policy mypolicy --set-priority -500
```

In the above example `-500` is a lower priority value but has higher precedence. Thus, `-500` will execute before `-100`.

Lower numerical priority values have higher precedence and are applied first.

1.9.3. Using policy objects to filter traffic between locally hosted containers and a network physically connected to the host

The policy objects feature allows users to filter traffic between Podman and firewalld zones.



NOTE

Red Hat recommends blocking all traffic by default and opening the selective services needed for the Podman utility.

Procedure

1. Create a new firewall policy:

```
# firewall-cmd --permanent --new-policy podmanToAny
```

2. Block all traffic from Podman to other zones and allow only necessary services on Podman:

```
# firewall-cmd --permanent --policy podmanToAny --set-target REJECT
# firewall-cmd --permanent --policy podmanToAny --add-service dhcp
# firewall-cmd --permanent --policy podmanToAny --add-service dns
# firewall-cmd --permanent --policy podmanToAny --add-service https
```

3. Create a new Podman zone:

```
# firewall-cmd --permanent --new-zone=podman
```

4. Define the ingress zone for the policy:

```
# firewall-cmd --permanent --policy podmanToHost --add-ingress-zone podman
```

5. Define the egress zone for all other zones:

```
# firewall-cmd --permanent --policy podmanToHost --add-egress-zone ANY
```

Setting the egress zone to ANY means that you filter from Podman to other zones. If you want to filter to the host, then set the egress zone to HOST.

6. Restart the firewalld service:

```
# systemctl restart firewalld
```

Verification

- Verify the Podman firewall policy to other zones:

```
# firewall-cmd --info-policy podmanToAny
podmanToAny (active)
...
target: REJECT
ingress-zones: podman
egress-zones: ANY
services: dhcp dns https
...
```

1.9.4. Setting the default target of policy objects

You can specify `--set-target` options for policies. The following targets are available:

- **ACCEPT** - accepts the packet
- **DROP** - drops the unwanted packets
- **REJECT** - rejects unwanted packets with an ICMP reply
- **CONTINUE** (default) - packets will be subject to rules in following policies and zones.

```
# firewall-cmd --permanent --policy mypolicy --set-target CONTINUE
```

Verification

- Verify information about the policy

```
# firewall-cmd --info-policy mypolicy
```

1.10. CONFIGURING NAT USING FIREWALLD

With **firewalld**, you can configure the following network address translation (NAT) types:

- Masquerading
- Destination NAT (DNAT)
- Redirect

1.10.1. Network address translation types

These are the different network address translation (NAT) types:

Masquerading

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers (ISPs) do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the internet, map the source IP address of packets from these ranges to a public IP address.

Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to a different port on the local machine. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

1.10.2. Configuring IP address masquerading

You can enable IP masquerading on your system. IP masquerading hides individual machines behind a gateway when accessing the internet.

Procedure

1. To check if IP masquerading is enabled (for example, for the **external** zone), enter the following command as **root**:

```
# firewall-cmd --zone=external --query-masquerade
```

The command prints **yes** with exit status **0** if enabled. It prints **no** with exit status **1** otherwise. If **zone** is omitted, the default zone will be used.

2. To enable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --add-masquerade
```

3. To make this setting persistent, pass the **--permanent** option to the command.
4. To disable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --remove-masquerade
```

To make this setting permanent, pass the **--permanent** option to the command.

1.10.3. Using DNAT to forward incoming HTTP traffic

You can use destination network address translation (DNAT) to direct incoming traffic from one destination address and port to another. Typically, this is useful for redirecting incoming requests from an external network interface to specific internal servers or services.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Forward incoming HTTP traffic:

```
# firewall-cmd --zone=public --add-forward-  
port=port=80:proto=tcp:toaddr=198.51.100.10:toport=8080 --permanent
```

The previous command defines a DNAT rule with the following settings:

- **--zone=public** - The firewall zone for which you configure the DNAT rule. You can adjust this to whatever zone you need.
 - **--add-forward-port** - The option that indicates you are adding a port-forwarding rule.
 - **port=80** - The external destination port.
 - **proto=tcp** - The protocol indicating that you forward TCP traffic.
 - **toaddr=198.51.100.10** - The destination IP address.
 - **toport=8080** - The destination port of the internal server.
 - **--permanent** - The option that makes the DNAT rule persistent across reboots.
2. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

- Verify the DNAT rule for the firewall zone that you used:

```
# firewall-cmd --list-forward-ports --zone=public  
port=80:proto=tcp:toport=8080:toaddr=198.51.100.10
```

Alternatively, view the corresponding XML configuration file:

```
# cat /etc/firewalld/zones/public.xml  
<?xml version="1.0" encoding="utf-8"?>  
<zone>  
  <short>Public</short>  
  <description>For use in public areas. You do not trust the other computers on networks to  
not harm your computer. Only selected incoming connections are accepted.</description>  
  <service name="ssh"/>  
  <service name="dhcpv6-client"/>  
</zone>
```



```
<service name="cockpit"/>
<forward-port port="80" protocol="tcp" to-port="8080" to-addr="198.51.100.10"/>
<forward/>
</zone>
```

Additional resources

- [Configuring kernel parameters at runtime](#)
- **firewall-cmd(1)** manual page

1.10.4. Redirecting traffic from a non-standard port to make the web service accessible on a standard port

You can use the redirect mechanism to make the web service that internally runs on a non-standard port accessible without requiring users to specify the port in the URL. As a result, the URLs are simpler and provide better browsing experience, while a non-standard port is still used internally or for specific requirements.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Create the NAT redirect rule:

```
# firewall-cmd --zone=public --add-forward-
port=port=<standard_port>:proto=tcp:toport=<non_standard_port> --permanent
```

The previous command defines the NAT redirect rule with the following settings:

- **--zone=public** - The firewall zone, for which you configure the rule. You can adjust this to whatever zone you need.
 - **--add-forward-port=port=<non_standard_port>** - The option that indicates you are adding a port-forwarding (redirecting) rule with source port on which you initially receive the incoming traffic.
 - **proto=tcp** - The protocol indicating that you redirect TCP traffic.
 - **toport=<standard_port>** - The destination port, to which the incoming traffic should be redirected after being received on the source port.
 - **--permanent** - The option that makes the rule persist across reboots.
2. Reload the firewall configuration to apply the changes:

```
# firewall-cmd --reload
```

Verification

- Verify the redirect rule for the firewall zone that you used:

```
# firewall-cmd --list-forward-ports
port=8080:proto=tcp:toport=80:toaddr=
```

Alternatively, view the corresponding XML configuration file:

```
# cat /etc/firewalld/zones/public.xml
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Public</short>
  <description>For use in public areas. You do not trust the other computers on networks to
not harm your computer. Only selected incoming connections are accepted.</description>
  <service name="ssh"/>
  <service name="dhcpv6-client"/>
  <service name="cockpit"/>
  <forward-port port="8080" protocol="tcp" to-port="80"/>
  <forward/>
</zone>
```

Additional resources

- [Configuring kernel parameters at runtime](#)
- **firewall-cmd(1)** manual page

1.11. PRIORITIZING RICH RULES

Rich rules provide a more advanced and flexible way to define firewall rules. Rich rules are particularly useful where services, ports, and so on are not enough to express complex firewall rules.

Concepts behind rich rules:

granularity and flexibility

You can define detailed conditions for network traffic based on more specific criteria.

rule structure

A rich rule consists of a family (IPv4 or IPv6), followed by conditions and actions.

```
rule family="ipv4|ipv6" [conditions] [actions]
```

conditions

They allow rich rules to apply only when certain criteria are met.

actions

You can define what happens to network traffic that matches the conditions.

combining multiple conditions

You can create more specific and complex filtering.

hierarchical control and reusability

You can combine rich rules with other firewall mechanisms such as zones or services.

By default, rich rules are organized based on their rule action. For example, **deny** rules have precedence over **allow** rules. The **priority** parameter in rich rules provides administrators fine-grained control over rich rules and their execution order. When using the **priority** parameter, rules are sorted first by their

priority values in ascending order. When more rules have the same **priority**, their order is determined by the rule action, and if the action is also the same, the order may be undefined.

1.11.1. How the priority parameter organizes rules into different chains

You can set the **priority** parameter in a rich rule to any number between **-32768** and **32767**, and lower numerical values have higher precedence.

The **firewalld** service organizes rules based on their priority value into different chains:

- Priority lower than 0: the rule is redirected into a chain with the **_pre** suffix.
- Priority higher than 0: the rule is redirected into a chain with the **_post** suffix.
- Priority equals 0: based on the action, the rule is redirected into a chain with the **_log**, **_deny**, or **_allow** the action.

Inside these sub-chains, **firewalld** sorts the rules based on their priority value.

Additional resources

- ``firewalld.richlanguage(5)`

1.11.2. Setting the priority of a rich rule

The following is an example of how to create a rich rule that uses the **priority** parameter to log all traffic that is not allowed or denied by other rules. You can use this rule to flag unexpected traffic.

Procedure

- Add a rich rule with a very low precedence to log all traffic that has not been matched by other rules:

```
# firewall-cmd --add-rich-rule='rule priority=32767 log prefix="UNEXPECTED: " limit
value="5/m"'
```

The command additionally limits the number of log entries to **5** per minute.

Verification

- Display the **nftables** rule that the command in the previous step created:

```
# nft list chain inet firewalld filter_IN_public_post
table inet firewalld {
  chain filter_IN_public_post {
    log prefix "UNEXPECTED: " limit rate 5/minute
  }
}
```

Additional resources

- ``firewalld.richlanguage(5)`

1.12. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE

Intra-zone forwarding is a **firewalld** feature that enables traffic forwarding between interfaces or sources within a **firewalld** zone.

1.12.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT

With intra-zone forwarding enabled, the traffic within a single **firewalld** zone can flow from one interface or source to another interface or source. The zone specifies the trust level of interfaces and sources. If the trust level is the same, the traffic stays inside the same zone.



NOTE

Enabling intra-zone forwarding in the default zone of **firewalld**, applies only to the interfaces and sources added to the current default zone.

firewalld uses different zones to manage incoming and outgoing traffic. Each zone has its own set of rules and behaviors. For example, the **trusted** zone, allows all forwarded traffic by default.

Other zones can have different default behaviors. In standard zones, forwarded traffic is typically dropped by default when the target of the zone is set to **default**.

To control how the traffic is forwarded between different interfaces or sources within a zone, make sure you understand and configure the target of the zone accordingly.

1.12.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network

You can use intra-zone forwarding to forward traffic between interfaces and sources within the same **firewalld** zone. This feature brings the following benefits:

- Seamless connectivity between wired and wireless devices (you can forward traffic between an Ethernet network connected to **enp1s0** and a Wi-Fi network connected to **wlp0s20**)
- Support for flexible work environments
- Shared resources that are accessible and used by multiple devices or users within a network (such as printers, databases, network-attached storage, and others)
- Efficient internal networking (such as smooth communication, reduced latency, resource accessibility, and others)

You can enable this functionality for individual **firewalld** zones.

Procedure

1. Enable packet forwarding in the kernel:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. Ensure that interfaces between which you want to enable intra-zone forwarding are assigned only to the **internal** zone:

```
# firewall-cmd --get-active-zones
```

3. If the interface is currently assigned to a zone other than **internal**, reassign it:

```
# firewall-cmd --zone=internal --change-interface=interface_name --permanent
```

4. Add the **enp1s0** and **wlp0s20** interfaces to the **internal** zone:

```
# firewall-cmd --zone=internal --add-interface=enp1s0 --add-interface=wlp0s20
```

5. Enable intra-zone forwarding:

```
# firewall-cmd --zone=internal --add-forward
```

Verification

The following Verification require that the **nmap-ncat** package is installed on both hosts.

1. Log in to a host that is on the same network as the **enp1s0** interface of the host on which you enabled zone forwarding.
2. Start an echo service with **ncat** to test connectivity:

```
# ncat -e /usr/bin/cat -l 12345
```

3. Log in to a host that is in the same network as the **wlp0s20** interface.
4. Connect to the echo server running on the host that is in the same network as the **enp1s0**:

```
# ncat <other_host> 12345
```

5. Type something and press **Enter**. Verify the text is sent back.

Additional resources

- **firewalld.zones(5)** man page on your system

1.13. CONFIGURING FIREWALLD BY USING RHEL SYSTEM ROLES

RHEL system roles is a set of contents for the Ansible automation utility. This content together with the Ansible automation utility provides a consistent configuration interface to remotely manage multiple systems at once.

The **rhel-system-roles** package contains the **rhel-system-roles.firewall** RHEL system role. This role was introduced for automated configurations of the **firewalld** service.

With the **firewall** RHEL system role you can configure many different **firewalld** parameters, for example:

- Zones

- The services for which packets should be allowed
- Granting, rejection, or dropping of traffic access to ports
- Forwarding of ports or port ranges for a zone

1.13.1. Resetting the firewalld settings by using the firewall RHEL system role

Over time, updates to your firewall configuration can accumulate to the point, where they could lead to unintended security risks. With the **firewall** RHEL system role, you can reset the **firewalld** settings to their default state in an automated fashion. This way you can efficiently remove any unintentional or insecure firewall rules and simplify their management.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Reset firewalld example
  hosts: managed-node-01.example.com
  tasks:
    - name: Reset firewalld
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
  vars:
    firewall:
      - previous: replaced
```

The settings specified in the example playbook include the following:

previous: replaced

Removes all existing user-defined settings and resets the **firewalld** settings to defaults. If you combine the **previous:replaced** parameter with other settings, the **firewall** role removes all existing settings before applying new ones.

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.firewall/README.md` file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- Run this command on the control node to remotely check that all firewall configuration on your managed node was reset to its default values:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd --list-all-zones'
```

Additional resources

- `/usr/share/ansible/roles/rhel-system-roles.firewall/README.md` file
- `/usr/share/doc/rhel-system-roles/firewall/` directory

1.13.2. Forwarding incoming traffic in firewalld from one local port to a different local port by using the firewall RHEL system role

You can use the **firewall** RHEL system role to remotely configure forwarding of incoming traffic from one local port to a different local port.

For example, if you have an environment where multiple services co-exist on the same machine and need the same default port, there are likely to become port conflicts. These conflicts can disrupt services and cause a downtime. With the **firewall** RHEL system role, you can efficiently forward traffic to alternative ports to ensure that your services can run simultaneously without modification to their configuration.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Forward incoming traffic on port 8080 to 443
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          - forward_port: 8080/tcp;443;
            state: enabled
            runtime: true
            permanent: true
```

The settings specified in the example playbook include the following:

forward_port: 8080/tcp;443

Traffic coming to the local port 8080 using the TCP protocol is forwarded to the port 443.

runtime: true

Enables changes in the runtime configuration. The default is set to **true**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On the control node, run the following command to remotely check the forwarded-ports on your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd  
--list-forward-ports'  
managed-node-01.example.com | CHANGED | rc=0 >>  
port=8080:proto=tcp:toport=443:toaddr=
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

1.13.3. Configuring a firewalld DMZ zone by using the firewall RHEL system role

As a system administrator, you can use the **firewall** RHEL system role to configure a **dmz** zone on the **enp1s0** interface to permit **HTTPS** traffic to the zone. In this way, you enable external users to access your web servers.

Prerequisites

- [You have prepared the control node and the managed nodes](#)
- You are logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.

Procedure

1. Create a playbook file, for example `~/playbook.yml`, with the following content:

```
---
- name: Configure firewalld
  hosts: managed-node-01.example.com
  tasks:
    - name: Creating a DMZ with access to HTTPS port and masquerading for hosts in DMZ
      ansible.builtin.include_role:
        name: redhat.rhel_system_roles.firewall
      vars:
        firewall:
          - zone: dmz
            interface: enp1s0
            service: https
            state: enabled
            runtime: true
            permanent: true
```

For details about all variables used in the playbook, see the `/usr/share/ansible/roles/rhel-system-roles.firewall/README.md` file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

- On the control node, run the following command to remotely check the information about the **dmz** zone on your managed node:

```
# ansible managed-node-01.example.com -m ansible.builtin.command -a 'firewall-cmd
--zone=dmz --list-all'
managed-node-01.example.com | CHANGED | rc=0 >>
dmz (active)
  target: default
  icmp-block-inversion: no
interfaces: enp1s0
  sources:
services: https ssh
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
```

Additional resources

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.firewall/README.md** file
- **/usr/share/doc/rhel-system-roles/firewall/** directory

CHAPTER 2. GETTING STARTED WITH NFTABLES

The **nftables** framework classifies packets and it is the successor to the **iptables**, **ip6tables**, **arptables**, **ebtables**, and **ipset** utilities. It offers numerous improvements in convenience, features, and performance over previous packet-filtering tools, most notably:

- Built-in lookup tables instead of linear processing
- A single framework for both the **IPv4** and **IPv6** protocols
- All rules applied atomically instead of fetching, updating, and storing a complete rule set
- Support for debugging and tracing in the rule set (**nfttrace**) and monitoring trace events (in the **nft** tool)
- More consistent and compact syntax, no protocol-specific extensions
- A Netlink API for third-party applications

The **nftables** framework uses tables to store chains. The chains contain individual rules for performing actions. The **nft** utility replaces all tools from the previous packet-filtering frameworks. You can use the **libnftnl** library for low-level interaction with **nftables** Netlink API through the **libmnl** library.

To display the effect of rule set changes, use the **nft list ruleset** command. Because these utilities add tables, chains, rules, sets, and other objects to the **nftables** rule set, be aware that **nftables** rule-set operations, such as the **nft flush ruleset** command, might affect rule sets installed using the **iptables** command.

2.1. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES

You can display **nftables** rule sets and manage them.

2.1.1. Basics of nftables tables

A table in **nftables** is a namespace that contains a collection of chains, rules, sets, and other objects.

Each table must have an address family assigned. The address family defines the packet types that this table processes. You can set one of the following address families when you create a table:

- **ip**: Matches only IPv4 packets. This is the default if you do not specify an address family.
- **ip6**: Matches only IPv6 packets.
- **inet**: Matches both IPv4 and IPv6 packets.
- **arp**: Matches IPv4 address resolution protocol (ARP) packets.
- **bridge**: Matches packets that pass through a bridge device.
- **netdev**: Matches packets from ingress.

If you want to add a table, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
}
```

- In shell scripts, use:

```
nft add table <table_address_family> <table_name>
```

2.1.2. Basics of nftables chains

Tables consist of chains which in turn are containers for rules. The following two rule types exists:

- **Base chain:** You can use base chains as an entry point for packets from the networking stack.
- **Regular chain:** You can use regular chains as a **jump** target to better organize rules.

If you want to add a base chain to a table, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
  chain <chain_name> {
    type <type> hook <hook> priority <priority>
    policy <policy> ;
  }
}
```

- In shell scripts, use:

```
nft add chain <table_address_family> <table_name> <chain_name> { type <type> hook
<hook> priority <priority> \; policy <policy> \; }
```

To avoid that the shell interprets the semicolons as the end of the command, place the `\` escape character in front of the semicolons.

Both examples create **base chains**. To create a **regular chain**, do not set any parameters in the curly brackets.

Chain types

The following are the chain types and an overview with which address families and hooks you can use them:

Type	Address families	Hooks	Description
filter	all	all	Standard chain type
nat	ip, ip6, inet	prerouting, input, output, postrouting	Chains of this type perform native address translation based on connection tracking entries. Only the first packet traverses this chain type.

Type	Address families	Hooks	Description
route	ip, ip6	output	Accepted packets that traverse this chain type cause a new route lookup if relevant parts of the IP header have changed.

Chain priorities

The priority parameter specifies the order in which packets traverse chains with the same hook value. You can set this parameter to an integer value or use a standard priority name.

The following matrix is an overview of the standard priority names and their numeric values, and with which address families and hooks you can use them:

Textual value	Numeric value	Address families	Hooks
raw	-300	ip, ip6, inet	all
mangle	-150	ip, ip6, inet	all
dstnat	-100	ip, ip6, inet	prerouting
	-300	bridge	prerouting
filter	0	ip, ip6, inet, arp, netdev	all
	-200	bridge	all
security	50	ip, ip6, inet	all
srcnat	100	ip, ip6, inet	postrouting
	300	bridge	postrouting
out	100	bridge	output

Chain policies

The chain policy defines whether **nftables** should accept or drop packets if rules in this chain do not specify any action. You can set one of the following policies in a chain:

- **accept** (default)
- **drop**

2.1.3. Basics of nftables rules

Rules define actions to perform on packets that pass a chain that contains this rule. If the rule also contains matching expressions, **nftables** performs the actions only if all previous expressions apply.

If you want to add a rule to a chain, the format to use depends on your firewall script:

- In scripts in native syntax, use:

```
table <table_address_family> <table_name> {
    chain <chain_name> {
        type <type> hook <hook> priority <priority> ; policy <policy> ;
        <rule>
    }
}
```

- In shell scripts, use:

```
nft add rule <table_address_family> <table_name> <chain_name> <rule>
```

This shell command appends the new rule at the end of the chain. If you prefer to add a rule at the beginning of the chain, use the **nft insert** command instead of **nft add**.

2.1.4. Managing tables, chains, and rules using nft commands

To manage an **nftables** firewall on the command line or in shell scripts, use the **nft** utility.



IMPORTANT

The commands in this procedure do not represent a typical workflow and are not optimized. This procedure only demonstrates how to use **nft** commands to manage tables, chains, and rules in general.

Procedure

1. Create a table named **nftables_svc** with the **inet** address family so that the table can process both IPv4 and IPv6 packets:

```
# nft add table inet nftables_svc
```

2. Add a base chain named **INPUT**, that processes incoming network traffic, to the **inet nftables_svc** table:

```
# nft add chain inet nftables_svc INPUT { type filter hook input priority filter\; policy accept\; }
```

To avoid that the shell interprets the semicolons as the end of the command, escape the semicolons using the **** character.

3. Add rules to the **INPUT** chain. For example, allow incoming TCP traffic on port 22 and 443, and, as the last rule of the **INPUT** chain, reject other incoming traffic with an Internet Control Message Protocol (ICMP) port unreachable message:

```
# nft add rule inet nftables_svc INPUT tcp dport 22 accept  
# nft add rule inet nftables_svc INPUT tcp dport 443 accept  
# nft add rule inet nftables_svc INPUT reject with icmpx type port-unreachable
```

If you enter the **nft add rule** commands as shown, **nft** adds the rules in the same order to the chain as you run the commands.

4. Display the current rule set including handles:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 443 accept # handle 3
    reject # handle 4
  }
}
```

5. Insert a rule before the existing rule with handle 3. For example, to insert a rule that allows TCP traffic on port 636, enter:

```
# nft insert rule inet nftables_svc INPUT position 3 tcp dport 636 accept
```

6. Append a rule after the existing rule with handle 3. For example, to insert a rule that allows TCP traffic on port 80, enter:

```
# nft add rule inet nftables_svc INPUT position 3 tcp dport 80 accept
```

7. Display the rule set again with handles. Verify that the later added rules have been added to the specified positions:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    tcp dport 80 accept # handle 6
    reject # handle 4
  }
}
```

8. Remove the rule with handle 6:

```
# nft delete rule inet nftables_svc INPUT handle 6
```

To remove a rule, you must specify the handle.

9. Display the rule set, and verify that the removed rule is no longer present:

```
# nft -a list table inet nftables_svc
table inet nftables_svc { # handle 13
  chain INPUT { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    reject # handle 4
  }
}
```

10. Remove all remaining rules from the **INPUT** chain:

```
# nft flush chain inet nftables_svc INPUT
```

11. Display the rule set, and verify that the **INPUT** chain is empty:

```
# nft list table inet nftables_svc
table inet nftables_svc {
  chain INPUT {
    type filter hook input priority filter; policy accept
  }
}
```

12. Delete the **INPUT** chain:

```
# nft delete chain inet nftables_svc INPUT
```

You can also use this command to delete chains that still contain rules.

13. Display the rule set, and verify that the **INPUT** chain has been deleted:

```
# nft list table inet nftables_svc
table inet nftables_svc {
}
```

14. Delete the **nftables_svc** table:

```
# nft delete table inet nftables_svc
```

You can also use this command to delete tables that still contain chains.



NOTE

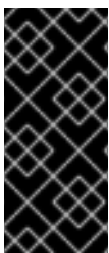
To delete the entire rule set, use the **nft flush ruleset** command instead of manually deleting all rules, chains, and tables in separate commands.

Additional resources

nft(8) man page on your system

2.2. MIGRATING FROM IPTABLES TO NFTABLES

If your firewall configuration still uses **iptables** rules, you can migrate your **iptables** rules to **nftables**.



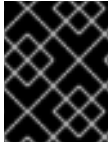
IMPORTANT

The **ipset** and **iptables-nft** packages have been deprecated in Red Hat Enterprise Linux 9. This includes deprecation of **nft-variants** such as **iptables**, **ip6tables**, **arptables**, and **ebtables** utilities. If you are using any of these tools, for example, because you upgraded from an earlier RHEL version, Red Hat recommends migrating to the **nft** command line tool provided by the **nftables** package.

2.2.1. When to use firewalld, or nftables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance-critical firewalls, such as for a whole network.



IMPORTANT

To prevent the different firewall-related services (**firewalld**, or **nftables**) from influencing each other, run only one of them on a RHEL host, and disable the other services.

2.2.2. Concepts in the nftables framework

Compared to the **iptables** framework, **nftables** offers a more modern, efficient, and flexible alternative. There are several concepts and features that provide advanced capabilities and improvements over **iptables**. These enhancements simplify the rule management and improve performance to make **nftables** a modern alternative for complex and high-performance networking environments.

The **nftables** framework contains the following components:

Tables and namespaces

In **nftables**, tables represent organizational units or namespaces that group together related firewall chains, sets, flowtables, and other objects. In **nftables**, tables provide a more flexible way to structure firewall rules and related components. While in **iptables**, the tables were more rigidly defined with specific purposes.

Table families

Each table in **nftables** is associated with a specific family (**ip**, **ip6**, **inet**, **arp**, **bridge**, or **netdev**). This association determines which packets the table can process. For example, a table in the **ip** family handles only IPv4 packets. On the other hand, **inet** is a special case of table family. It offers a unified approach across protocols, because it can process both IPv4 and IPv6 packets. Another case of a special table family is **netdev**, because it is used for rules that apply directly to network devices, enabling filtering at the device level.

Base chains

Base chains in **nftables** are highly configurable entry-points in the packet processing pipeline that enable users to specify the following:

- Type of chain, for example "filter"
- The hook point in the packet processing path, for example "input", "output", "forward"
- Priority of the chain

This flexibility enables precise control over when and how the rules are applied to packets as they pass through the network stack. A special case of a chain is the **route** chain, which is used to influence the routing decisions made by the kernel, based on packet headers.

Virtual machine for rule processing

The **nftables** framework uses an internal virtual machine to process rules. This virtual machine executes instructions that are similar to assembly language operations (loading data into registers, performing comparisons, and so on). Such a mechanism allows for highly flexible and efficient rule

processing.

Enhancements in **nftables** can be introduced as new instructions for that virtual machine. This typically requires a new kernel module and updates to the **libnftnl** library and the **nft** command-line utility.

Alternatively, you can introduce new features by combining existing instructions in innovative ways without a need for kernel modifications. The syntax of **nftables** rules reflects the flexibility of the underlying virtual machine. For example, the rule **meta mark set tcp dport map { 22: 1, 80: 2 }** sets a packet's firewall mark to 1 if the TCP destination port is 22, and to 2 if the port is 80. This demonstrates how complex logic can be expressed concisely.

Lessons learned and enhancements

The **nftables** framework integrates and extends the functionality of the **ipset** utility, which is used in **iptables** for bulk matching on IP addresses, ports, other data types and, most importantly, combinations thereof. This integration makes it easier to manage large and dynamic sets of data directly within **nftables**. Next, **nftables** natively supports matching packets based on multiple values or ranges for any data type, which enhances its capability to handle complex filtering requirements. With **nftables** you can manipulate any field within a packet.

In **nftables**, sets can be either named or anonymous. The named sets can be referenced by multiple rules and modified dynamically. The anonymous sets are defined inline within a rule and are immutable. Sets can contain elements that are combinations of different types, for example IP address and port number pairs. This feature provides greater flexibility in matching complex criteria. To manage sets, the kernel can select the most appropriate backend based on the specific requirements (performance, memory efficiency, and others). Sets can also function as maps with key-value pairs. The value part can be used as data points (values to write into packet headers), or as verdicts or chains to jump to. This enables complex and dynamic rule behaviors, known as "verdict maps".

Flexible rule format

The structure of **nftables** rules is straightforward. The conditions and actions are applied sequentially from left to right. This intuitive format simplifies rule creating and troubleshooting.

Conditions in a rule are logically connected (with the AND operator) together, which means that all conditions must be evaluated as "true" for the rule to match. If any condition fails, the evaluation moves to the next rule.

Actions in **nftables** can be final, such as **drop** or **accept**, which stop further rule processing for the packet. Non-terminal actions, such as **counter log meta mark set 0x3**, perform specific tasks (counting packets, logging, setting a mark, and others), but allow subsequent rules to be evaluated.

Additional resources

- **nft(8)** man page
- [What comes after iptables? Its successor, of course: nftables](#)
- [Firewalld: The Future is nftables](#)

2.2.3. Concepts in the deprecated iptables framework

Similar to the actively-maintained **nftables** framework, the deprecated **iptables** framework enables you to perform a variety of packet filtering tasks, logging and auditing, NAT-related configuration tasks, and more.

The **iptables** framework is structured into multiple tables, where each table is designed for a specific purpose:

filter

The default table, ensures general packet filtering

nat

For Network Address Translation (NAT), includes altering the source and destination addresses of packets

mangle

For specific packet alteration, enables you to do modification of packet headers for advanced routing decisions

raw

For configurations that need to happen before connection tracking

These tables are implemented as separate kernel modules, where each table offers a fixed set of builtin chains such as **INPUT**, **OUTPUT**, and **FORWARD**. A chain is a sequence of rules that packets are evaluated against. These chains hook into specific points in the packet processing flow in the kernel. The chains have the same names across different tables, however their order of execution is determined by their respective hook priorities. The priorities are managed internally by the kernel to make sure that the rules are applied in the correct sequence.

Originally, **iptables** was designed to process IPv4 traffic. However, with the inception of the IPv6 protocol, the **ip6tables** utility needed to be introduced to provide comparable functionality (as **iptables**) and enable users to create and manage firewall rules for IPv6 packets. With the same logic, the **arptables** utility was created to process Address Resolution Protocol (ARP) and the **ebtables** utility was developed to handle Ethernet bridging frames. These tools ensure that you can apply the packet filtering abilities of **iptables** across various network protocols and provide comprehensive network coverage.

To enhance the functionality of **iptables**, the extensions started to be developed. The functionality extensions are typically implemented as kernel modules that are paired with user-space dynamic shared objects (DSOs). The extensions introduce "matches" and "targets" that you can use in firewall rules to perform more sophisticated operations. Extensions can enable complex matches and targets. For instance you can match on, or manipulate specific layer 4 protocol header values, perform rate-limiting, enforce quotas, and so on. Some extensions are designed to address limitations in the default **iptables** syntax, for example the "multiport" match extension. This extension allows a single rule to match multiple, non-consecutive ports to simplify rule definitions, and thereby reducing the number of individual rules required.

An **ipset** is a special kind of functionality extension to **iptables**. It is a kernel-level data structure that is used together with **iptables** to create collections of IP addresses, port numbers, and other network-related elements that you can match against packets. These sets significantly streamline, optimize, and accelerate the process of writing and managing firewall rules.

Additional resources

- **iptables(8)** man page

2.2.4. Converting iptables and ip6tables rule sets to nftables

Use the **iptables-restore-translate** and **ip6tables-restore-translate** utilities to translate **iptables** and **ip6tables** rule sets to **nftables**.

Prerequisites

- The **nftables** and **iptables** packages are installed.

- The system has **iptables** and **ip6tables** rules configured.

Procedure

1. Write the **iptables** and **ip6tables** rules to a file:

```
# iptables-save >/root/iptables.dump
# ip6tables-save >/root/ip6tables.dump
```

2. Convert the dump files to **nftables** instructions:

```
# iptables-restore-translate -f /root/iptables.dump > /etc/nftables/ruleset-migrated-
from-iptables.nft
# ip6tables-restore-translate -f /root/ip6tables.dump > /etc/nftables/ruleset-migrated-
from-ip6tables.nft
```

3. Review and, if needed, manually update the generated **nftables** rules.
4. To enable the **nftables** service to load the generated files, add the following to the **/etc/sysconfig/nftables.conf** file:

```
include "/etc/nftables/ruleset-migrated-from-iptables.nft"
include "/etc/nftables/ruleset-migrated-from-ip6tables.nft"
```

5. Stop and disable the **iptables** service:

```
# systemctl disable --now iptables
```

If you used a custom script to load the **iptables** rules, ensure that the script no longer starts automatically and reboot to flush all tables.

6. Enable and start the **nftables** service:

```
# systemctl enable --now nftables
```

Verification

- Display the **nftables** rule set:

```
# nft list ruleset
```

Additional resources

- [Automatically loading nftables rules when the system boots](#)

2.2.5. Converting single iptables and ip6tables rules to nftables

Red Hat Enterprise Linux provides the **iptables-translate** and **ip6tables-translate** utilities to convert an **iptables** or **ip6tables** rule into the equivalent one for **nftables**.

Prerequisites

- The **nftables** package is installed.

Procedure

- Use the **iptables-translate** or **ip6tables-translate** utility instead of **iptables** or **ip6tables** to display the corresponding **nftables** rule, for example:

```
# iptables-translate -A INPUT -s 192.0.2.0/24 -j ACCEPT
nft add rule ip filter INPUT ip saddr 192.0.2.0/24 counter accept
```

Note that some extensions lack translation support. In these cases, the utility prints the untranslated rule prefixed with the **#** sign, for example:

```
# iptables-translate -A INPUT -j CHECKSUM --checksum-fill
nft # -A INPUT -j CHECKSUM --checksum-fill
```

Additional resources

- **iptables-translate --help**

2.2.6. Comparison of common iptables and nftables commands

The following is a comparison of common **iptables** and **nftables** commands:

- Listing all rules:

iptables	nftables
iptables-save	nft list ruleset

- Listing a certain table and chain:

iptables	nftables
iptables -L	nft list table ip filter
iptables -L INPUT	nft list chain ip filter INPUT
iptables -t nat -L PREROUTING	nft list chain ip nat PREROUTING

The **nft** command does not pre-create tables and chains. They exist only if a user created them manually.

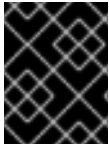
Listing rules generated by firewalld:

```
# nft list table inet firewalld
# nft list table ip firewalld
# nft list table ip6 firewalld
```

2.3. CONFIGURING NAT USING NFTABLES

With **nftables**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect



IMPORTANT

You can only use real interface names in **iifname** and **oifname** parameters, and alternative names (**altname**) are not supported.

2.3.1. NAT types

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers (ISPs) do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the internet, map the source IP address of packets from these ranges to a public IP address.

Masquerading and SNAT are very similar to one another. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

2.3.2. Configuring masquerading using nftables

Masquerading enables a router to dynamically change the source IP of packets sent through an interface to the IP address of the interface. This means that if the interface gets a new IP assigned, **nftables** automatically uses the new IP when replacing the source IP.

Replace the source IP of packets leaving the host through the **ens3** interface to the IP set on **ens3**.

Procedure

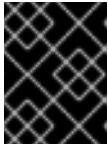
1. Create a table:

■

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```



IMPORTANT

Even if you do not add a rule to the **prerouting** chain, the **nftables** framework requires this chain to match incoming packet replies.

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that matches outgoing packets on the **ens3** interface:

```
# nft add rule nat postrouting oifname "ens3" masquerade
```

2.3.3. Configuring source NAT using nftables

On a router, Source NAT (SNAT) enables you to change the IP of packets sent through an interface to a specific IP address. The router then replaces the source IP of outgoing packets.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```



IMPORTANT

Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that replaces the source IP of outgoing packets through **ens3** with **192.0.2.1**:

```
# nft add rule nat postrouting oifname "ens3" snat to 192.0.2.1
```

2.3.4. Configuring destination NAT using nftables

Destination NAT (DNAT) enables you to redirect traffic on a router to a host that is not directly accessible from the internet.

For example, with DNAT the router redirects incoming traffic sent to port **80** and **443** to a web server with the IP address **192.0.2.1**.

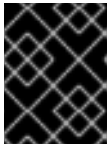
Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```



IMPORTANT

Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic to port **80** and **443** on the **ens3** interface of the router to the web server with the IP address **192.0.2.1**:

```
# nft add rule nat prerouting iifname ens3 tcp dport { 80, 443 } dnat to 192.0.2.1
```

4. Depending on your environment, add either a SNAT or masquerading rule to change the source address for packets returning from the web server to the sender:

- a. If the **ens3** interface uses a dynamic IP addresses, add a masquerading rule:

```
# nft add rule nat postrouting oifname "ens3" masquerade
```

- b. If the **ens3** interface uses a static IP address, add a SNAT rule. For example, if the **ens3** uses the **198.51.100.1** IP address:

```
# nft add rule nat postrouting oifname "ens3" snat to 198.51.100.1
```

5. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

Additional resources

- [NAT types](#)

2.3.5. Configuring a redirect using nftables

The **redirect** feature is a special case of destination network address translation (DNAT) that redirects packets to the local machine depending on the chain hook.

For example, you can redirect incoming and forwarded traffic sent to port **22** of the local host to port **2222**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** chain to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
```

Note that you must pass the **--** option to the **nft** command to prevent the shell from interpreting the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic on port **22** to port **2222**:

```
# nft add rule nat prerouting tcp dport 22 redirect to 2222
```

Additional resources

- [NAT types](#)

2.4. WRITING AND EXECUTING NFTABLES SCRIPTS

The major benefit of using the **nftables** framework is that the execution of scripts is atomic. This means that the system either applies the whole script or prevents the execution if an error occurs. This guarantees that the firewall is always in a consistent state.

Additionally, with the **nftables** script environment, you can:

- Add comments
- Define variables
- Include other rule-set files

When you install the **nftables** package, Red Hat Enterprise Linux automatically creates ***.nft** scripts in the **/etc/nftables/** directory. These scripts contain commands that create tables and empty chains for different purposes.

2.4.1. Supported nftables script formats

You can write scripts in the **nftables** scripting environment in the following formats:

- The same format as the **nft list ruleset** command displays the rule set:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset

table inet example_table {
```

```
chain example_chain {
    # Chain for incoming packets that drops all packets that
    # are not explicitly allowed by any rule in this chain
    type filter hook input priority 0; policy drop;

    # Accept connections to port 22 (ssh)
    tcp dport ssh accept
}
}
```

- The same syntax as for **nft** commands:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset

# Create a table
add table inet example_table

# Create a chain for incoming packets that drops all packets
# that are not explicitly allowed by any rule in this chain
add chain inet example_table example_chain { type filter hook input priority 0 ; policy drop ; }

# Add a rule that accepts connections to port 22 (ssh)
add rule inet example_table example_chain tcp dport ssh accept
```

2.4.2. Running nftables scripts

You can run an **nftables** script either by passing it to the **nft** utility or by executing the script directly.

Procedure

- To run an **nftables** script by passing it to the **nft** utility, enter:

```
# nft -f /etc/nftables/<example_firewall_script>.nft
```

- To run an **nftables** script directly:

a. For the single time that you perform this:

- i. Ensure that the script starts with the following shebang sequence:

```
#!/usr/sbin/nft -f
```



IMPORTANT

If you omit the **-f** parameter, the **nft** utility does not read the script and displays: **Error: syntax error, unexpected newline, expecting string.**

- ii. Optional: Set the owner of the script to **root**:

```
# chown root /etc/nftables/<example_firewall_script>.nft
```

iii. Make the script executable for the owner:

```
# chmod u+x /etc/nftables/<example_firewall_script>.nft
```

b. Run the script:

```
# /etc/nftables/<example_firewall_script>.nft
```

If no output is displayed, the system executed the script successfully.



IMPORTANT

Even if **nft** executes the script successfully, incorrectly placed rules, missing parameters, or other problems in the script can cause that the firewall behaves not as expected.

Additional resources

- **chown(1)** and **chmod(1)** man pages on your system
- [Automatically loading nftables rules when the system boots](#)

2.4.3. Using comments in nftables scripts

The **nftables** scripting environment interprets everything to the right of a **#** character to the end of a line as a comment.

Comments can start at the beginning of a line, or next to a command:

```
...
# Flush the rule set
flush ruleset

add table inet example_table # Create a table
...
```

2.4.4. Using variables in nftables script

To define a variable in an **nftables** script, use the **define** keyword. You can store single values and anonymous sets in a variable. For more complex scenarios, use sets or verdict maps.

Variables with a single value

The following example defines a variable named **INET_DEV** with the value **enp1s0**:

```
define INET_DEV = enp1s0
```

You can use the variable in the script by entering the **\$** sign followed by the variable name:

```
...
add rule inet example_table example_chain iifname $INET_DEV tcp dport ssh accept
...
```

Variables that contain an anonymous set

The following example defines a variable that contains an anonymous set:

```
define DNS_SERVERS = { 192.0.2.1, 192.0.2.2 }
```

You can use the variable in the script by writing the **\$** sign followed by the variable name:

```
add rule inet example_table example_chain ip daddr $DNS_SERVERS accept
```



NOTE

Curly braces have special semantics when you use them in a rule because they indicate that the variable represents a set.

Additional resources

- [Using sets in nftables commands](#)
- [Using verdict maps in nftables commands](#)

2.4.5. Including files in nftables scripts

In the **nftables** scripting environment, you can include other scripts by using the **include** statement.

If you specify only a file name without an absolute or relative path, **nftables** includes files from the default search path, which is set to **/etc** on Red Hat Enterprise Linux.

Example 2.1. Including files from the default search directory

To include a file from the default search directory:

```
include "example.nft"
```

Example 2.2. Including all *.nft files from a directory

To include all files ending with ***.nft** that are stored in the **/etc/nftables/rulesets/** directory:

```
include "/etc/nftables/rulesets/*.nft"
```

Note that the **include** statement does not match files beginning with a dot.

Additional resources

- The **Include files** section in the **nft(8)** man page on your system

2.4.6. Automatically loading nftables rules when the system boots

The **nftables** systemd service loads firewall scripts that are included in the **/etc/sysconfig/nftables.conf** file.

Prerequisites

- The **nftables** scripts are stored in the **/etc/nftables/** directory.

Procedure

1. Edit the **/etc/sysconfig/nftables.conf** file.

- If you modified the ***.nft** scripts that were created in **/etc/nftables/** with the installation of the **nftables** package, uncomment the **include** statement for these scripts.
- If you wrote new scripts, add **include** statements to include these scripts. For example, to load the **/etc/nftables/example.nft** script when the **nftables** service starts, add:

```
include "/etc/nftables/_example_.nft"
```

2. Optional: Start the **nftables** service to load the firewall rules without rebooting the system:

```
# systemctl start nftables
```

3. Enable the **nftables** service.

```
# systemctl enable nftables
```

Additional resources

- [Supported nftables script formats](#)

2.5. USING SETS IN NFTABLES COMMANDS

The **nftables** framework natively supports sets. You can use sets, for example, if a rule should match multiple IP addresses, port numbers, interfaces, or any other match criteria.

2.5.1. Using anonymous sets in nftables

An anonymous set contains comma-separated values enclosed in curly brackets, such as **{ 22, 80, 443 }**, that you use directly in a rule. You can use anonymous sets also for IP addresses and any other match criteria.

The drawback of anonymous sets is that if you want to change the set, you must replace the rule. For a dynamic solution, use named sets as described in [Using named sets in nftables](#).

Prerequisites

- The **example_chain** chain and the **example_table** table in the **inet** family exists.

Procedure

1. For example, to add a rule to **example_chain** in **example_table** that allows incoming traffic to port **22, 80, and 443**:

```
# nft add rule inet example_table example_chain tcp dport { 22, 80, 443 } accept
```

- Optional: Display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport { ssh, http, https } accept
  }
}
```

2.5.2. Using named sets in nftables

The **nftables** framework supports mutable named sets. A named set is a list or range of elements that you can use in multiple rules within a table. Another benefit over anonymous sets is that you can update a named set without replacing the rules that use the set.

When you create a named set, you must specify the type of elements the set contains. You can set the following types:

- **ipv4_addr** for a set that contains IPv4 addresses or ranges, such as **192.0.2.1** or **192.0.2.0/24**.
- **ipv6_addr** for a set that contains IPv6 addresses or ranges, such as **2001:db8:1::1** or **2001:db8:1::1/64**.
- **ether_addr** for a set that contains a list of media access control (MAC) addresses, such as **52:54:00:6b:66:42**.
- **inet_proto** for a set that contains a list of internet protocol types, such as **tcp**.
- **inet_service** for a set that contains a list of internet services, such as **ssh**.
- **mark** for a set that contains a list of packet marks. Packet marks can be any positive 32-bit integer value (**0** to **2147483647**).

Prerequisites

- The **example_chain** chain and the **example_table** table exists.

Procedure

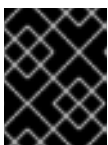
- Create an empty set. The following examples create a set for IPv4 addresses:

- To create a set that can store multiple individual IPv4 addresses:

```
# nft add set inet example_table example_set { type ipv4_addr \;
```

- To create a set that can store IPv4 address ranges:

```
# nft add set inet example_table example_set { type ipv4_addr \; flags interval \;
```



IMPORTANT

To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

- Optional: Create rules that use the set. For example, the following command adds a rule to the **example_chain** in the **example_table** that will drop all packets from IPv4 addresses in **example_set**.

```
# nft add rule inet example_table example_chain ip saddr @example_set drop
```

Because **example_set** is still empty, the rule has currently no effect.

- Add IPv4 addresses to **example_set**:

- If you create a set that stores individual IPv4 addresses, enter:

```
# nft add element inet example_table example_set { 192.0.2.1, 192.0.2.2 }
```

- If you create a set that stores IPv4 ranges, enter:

```
# nft add element inet example_table example_set { 192.0.2.0-192.0.2.255 }
```

When you specify an IP address range, you can alternatively use the Classless Inter-Domain Routing (CIDR) notation, such as **192.0.2.0/24** in the above example.

2.5.3. Using dynamic sets to add entries from the packet path

Dynamic sets in the **nftables** framework allow automatic addition of elements from packet data. For example, IP addresses, destination ports, MAC addresses, and others. This functionality enables you to collect those elements in real-time and use them to create deny lists, ban lists, and others so that you can instantly react to security threats.

Prerequisites

- The **example_chain** chain and the **example_table** table in the **inet** family exist.

Procedure

- Create an empty set. The following examples create a set for IPv4 addresses:

- To create a set that can store multiple individual IPv4 addresses:

```
# nft add set inet example_table example_set { type ipv4_addr \; }
```

- To create a set that can store IPv4 address ranges:

```
# nft add set inet example_table example_set { type ipv4_addr \; flags interval \; }
```



IMPORTANT

To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

- Create a rule for dynamically adding the source IPv4 addresses of incoming packets to the **example_set** set:

```
# nft add rule inet example_table example_chain set add ip saddr @example_set
```

The command creates a new rule within the **example_chain** rule chain and the **example_table** to dynamically add the source IPv4 address of the packet to the **example_set**.

Verification

- Ensure the rule was added:

```
# nft list ruleset
...
table ip example_table {
  set example_set {
    type ipv4_addr
    elements = { 192.0.2.250, 192.0.2.251 }
  }

  chain example_chain {
    type filter hook input priority 0
    add @example_set { ip saddr }
  }
}
```

The command displays the entire ruleset currently loaded in **nftables**. It shows that IPs are actively triggering the rule, and **example_set** is being updated with the relevant addresses.

Next steps

Once you have a dynamic set of IPs, you can use it for various security, filtering, and traffic control purposes. For example:

- block, limit, or log network traffic
- combine with allow-listing to avoid banning trusted users
- use automatic timeouts to prevent over-blocking

2.5.4. Additional resources

- The **Sets** section in the **nft(8)** man page on your system

2.6. USING VERDICT MAPS IN NFTABLES COMMANDS

Verdict maps, which are also known as dictionaries, enable **nft** to perform an action based on packet information by mapping match criteria to an action.

2.6.1. Using anonymous maps in nftables

An anonymous map is a **{ match_criteria : action }** statement that you use directly in a rule. The statement can contain multiple comma-separated mappings.

The drawback of an anonymous map is that if you want to change the map, you must replace the rule. For a dynamic solution, use named maps as described in [Using named maps in nftables](#).

For example, you can use an anonymous map to route both TCP and UDP packets of the IPv4 and IPv6 protocol to different chains to count incoming TCP and UDP packets separately.

Procedure

1. Create a new table:

```
# nft add table inet example_table
```

2. Create the **tcp_packets** chain in **example_table**:

```
# nft add chain inet example_table tcp_packets
```

3. Add a rule to **tcp_packets** that counts the traffic in this chain:

```
# nft add rule inet example_table tcp_packets counter
```

4. Create the **udp_packets** chain in **example_table**

```
# nft add chain inet example_table udp_packets
```

5. Add a rule to **udp_packets** that counts the traffic in this chain:

```
# nft add rule inet example_table udp_packets counter
```

6. Create a chain for incoming traffic. For example, to create a chain named **incoming_traffic** in **example_table** that filters incoming traffic:

```
# nft add chain inet example_table incoming_traffic { type filter hook input priority 0 \;  
}
```

7. Add a rule with an anonymous map to **incoming_traffic**:

```
# nft add rule inet example_table incoming_traffic ip protocol vmap { tcp : jump  
tcp_packets, udp : jump udp_packets }
```

The anonymous map distinguishes the packets and sends them to the different counter chains based on their protocol.

8. To list the traffic counters, display **example_table**:

```
# nft list table inet example_table  
table inet example_table {  
  chain tcp_packets {  
    counter packets 36379 bytes 2103816  
  }  
  
  chain udp_packets {  
    counter packets 10 bytes 1559  
  }  
  
  chain incoming_traffic {  
    type filter hook input priority filter; policy accept;  
    ip protocol vmap { tcp : jump tcp_packets, udp : jump udp_packets }  
  }  
}
```

The counters in the **tcp_packets** and **udp_packets** chain display both the number of received packets and bytes.

2.6.2. Using named maps in nftables

The **nftables** framework supports named maps. You can use these maps in multiple rules within a table. Another benefit over anonymous maps is that you can update a named map without replacing the rules that use it.

When you create a named map, you must specify the type of elements:

- **ipv4_addr** for a map whose match part contains an IPv4 address, such as **192.0.2.1**.
- **ipv6_addr** for a map whose match part contains an IPv6 address, such as **2001:db8:1::1**.
- **ether_addr** for a map whose match part contains a media access control (MAC) address, such as **52:54:00:6b:66:42**.
- **inet_proto** for a map whose match part contains an internet protocol type, such as **tcp**.
- **inet_service** for a map whose match part contains an internet services name port number, such as **ssh** or **22**.
- **mark** for a map whose match part contains a packet mark. A packet mark can be any positive 32-bit integer value (**0** to **2147483647**).
- **counter** for a map whose match part contains a counter value. The counter value can be any positive 64-bit integer value.
- **quota** for a map whose match part contains a quota value. The quota value can be any positive 64-bit integer value.

For example, you can allow or drop incoming packets based on their source IP address. Using a named map, you require only a single rule to configure this scenario while the IP addresses and actions are dynamically stored in the map.

Procedure

1. Create a table. For example, to create a table named **example_table** that processes IPv4 packets:

```
# nft add table ip example_table
```

2. Create a chain. For example, to create a chain named **example_chain** in **example_table**:

```
# nft add chain ip example_table example_chain { type filter hook input priority 0 \;
```



IMPORTANT

To prevent the shell from interpreting the semicolons as the end of the command, you must escape the semicolons with a backslash.

3. Create an empty map. For example, to create a map for IPv4 addresses:

```
# nft add map ip example_table example_map { type ipv4_addr : verdict \; }
```

4. Create rules that use the map. For example, the following command adds a rule to **example_chain** in **example_table** that applies actions to IPv4 addresses which are both defined in **example_map**:

```
# nft add rule example_table example_chain ip saddr vmap @example_map
```

5. Add IPv4 addresses and corresponding actions to **example_map**:

```
# nft add element ip example_table example_map { 192.0.2.1 : accept, 192.0.2.2 : drop }
```

This example defines the mappings of IPv4 addresses to actions. In combination with the rule created above, the firewall accepts packet from **192.0.2.1** and drops packets from **192.0.2.2**.

6. Optional: Enhance the map by adding another IP address and action statement:

```
# nft add element ip example_table example_map { 192.0.2.3 : accept }
```

7. Optional: Remove an entry from the map:

```
# nft delete element ip example_table example_map { 192.0.2.1 }
```

8. Optional: Display the rule set:

```
# nft list ruleset
table ip example_table {
  map example_map {
    type ipv4_addr : verdict
    elements = { 192.0.2.2 : drop, 192.0.2.3 : accept }
  }

  chain example_chain {
    type filter hook input priority filter; policy accept;
    ip saddr vmap @example_map
  }
}
```

2.6.3. Additional resources

- The **Maps** section in the **nft(8)** man page on your system

2.7. EXAMPLE: PROTECTING A LAN AND DMZ USING AN NFTABLES SCRIPT

Use the **nftables** framework on a RHEL router to write and install a firewall script that protects the network clients in an internal LAN and a web server in a DMZ from unauthorized access from the internet and from other networks.



IMPORTANT

This example is only for demonstration purposes and describes a scenario with specific requirements.

Firewall scripts highly depend on the network infrastructure and security requirements. Use this example to learn the concepts of **nftables** firewalls when you write scripts for your own environment.

2.7.1. Network conditions

The network in this example has the following conditions:

- The router is connected to the following networks:
 - The internet through interface **enp1s0**
 - The internal LAN through interface **enp7s0**
 - The DMZ through **enp8s0**
- The internet interface of the router has both a static IPv4 address (**203.0.113.1**) and IPv6 address (**2001:db8:a::1**) assigned.
- The clients in the internal LAN use only private IPv4 addresses from the range **10.0.0.0/24**. Consequently, traffic from the LAN to the internet requires source network address translation (SNAT).
- The administrator PCs in the internal LAN use the IP addresses **10.0.0.100** and **10.0.0.200**.
- The DMZ uses public IP addresses from the ranges **198.51.100.0/24** and **2001:db8:b::/56**.
- The web server in the DMZ uses the IP addresses **198.51.100.5** and **2001:db8:b::5**.
- The router acts as a caching DNS server for hosts in the LAN and DMZ.

2.7.2. Security requirements to the firewall script

The following are the requirements to the **nftables** firewall in the example network:

- The router must be able to:
 - Recursively resolve DNS queries.
 - Perform all connections on the loopback interface.
- Clients in the internal LAN must be able to:
 - Query the caching DNS server running on the router.
 - Access the HTTPS server in the DMZ.
 - Access any HTTPS server on the internet.
- The PCs of the administrators must be able to access the router and every server in the DMZ using SSH.

- The web server in the DMZ must be able to:
 - Query the caching DNS server running on the router.
 - Access HTTPS servers on the internet to download updates.
- Hosts on the internet must be able to:
 - Access the HTTPS servers in the DMZ.
- Additionally, the following security requirements exists:
 - Connection attempts that are not explicitly allowed should be dropped.
 - Dropped packets should be logged.

2.7.3. Configuring logging of dropped packets to a file

By default, **systemd** logs kernel messages, such as for dropped packets, to the journal. Additionally, you can configure the **rsyslog** service to log such entries to a separate file. To ensure that the log file does not grow infinitely, configure a rotation policy.

Prerequisites

- The **rsyslog** package is installed.
- The **rsyslog** service is running.

Procedure

1. Create the **/etc/rsyslog.d/nftables.conf** file with the following content:

```
:msg, startswith, "nft drop" -/var/log/nftables.log
& stop
```

Using this configuration, the **rsyslog** service logs dropped packets to the **/var/log/nftables.log** file instead of **/var/log/messages**.

2. Restart the **rsyslog** service:

```
# systemctl restart rsyslog
```

3. Create the **/etc/logrotate.d/nftables** file with the following content to rotate **/var/log/nftables.log** if the size exceeds 10 MB:

```
/var/log/nftables.log {
    size +10M
    maxage 30
    sharedscripts
    postrotate
        /usr/bin/systemctl kill -s HUP rsyslog.service >/dev/null 2>&1 || true
    endscript
}
```

The **maxage 30** setting defines that **logrotate** removes rotated logs older than 30 days during the next rotation operation.

Additional resources

- **rsyslog.conf(5)** and **logrotate(8)** man pages on your system

2.7.4. Writing and activating the nftables script

This example is an **nftables** firewall script that runs on a RHEL router and protects the clients in an internal LAN and a web server in a DMZ. For details about the network and the requirements for the firewall used in the example, see [Network conditions](#) and [Security requirements to the firewall script](#).



WARNING

This **nftables** firewall script is only for demonstration purposes. Do not use it without adapting it to your environments and security requirements.

Prerequisites

- The network is configured as described in [Network conditions](#).

Procedure

1. Create the **/etc/nftables/firewall.nft** script with the following content:

```
# Remove all rules
flush ruleset

# Table for both IPv4 and IPv6 rules
table inet nftables_svc {

    # Define variables for the interface name
    define INET_DEV = enp1s0
    define LAN_DEV = enp7s0
    define DMZ_DEV = enp8s0

    # Set with the IPv4 addresses of admin PCs
    set admin_pc_ipv4 {
        type ipv4_addr
        elements = { 10.0.0.100, 10.0.0.200 }
    }

    # Chain for incoming traffic. Default policy: drop
    chain INPUT {
        type filter hook input priority filter
        policy drop
    }
}
```

```

# Accept packets in established and related state, drop invalid packets
ct state vmap { established:accept, related:accept, invalid:drop }

# Accept incoming traffic on loopback interface
iifname lo accept

# Allow request from LAN and DMZ to local DNS server
iifname { $LAN_DEV, $DMZ_DEV } meta l4proto { tcp, udp } th dport 53 accept

# Allow admins PCs to access the router using SSH
iifname $LAN_DEV ip saddr @admin_pc_ipv4 tcp dport 22 accept

# Last action: Log blocked packets
# (packets that were not accepted in previous rules in this chain)
log prefix "nft drop IN : "
}

# Chain for outgoing traffic. Default policy: drop
chain OUTPUT {
    type filter hook output priority filter
    policy drop

    # Accept packets in established and related state, drop invalid packets
    ct state vmap { established:accept, related:accept, invalid:drop }

    # Accept outgoing traffic on loopback interface
    oifname lo accept

    # Allow local DNS server to recursively resolve queries
    oifname $INET_DEV meta l4proto { tcp, udp } th dport 53 accept

    # Last action: Log blocked packets
    log prefix "nft drop OUT: "
}

# Chain for forwarding traffic. Default policy: drop
chain FORWARD {
    type filter hook forward priority filter
    policy drop

    # Accept packets in established and related state, drop invalid packets
    ct state vmap { established:accept, related:accept, invalid:drop }

    # IPv4 access from LAN and internet to the HTTPS server in the DMZ
    iifname { $LAN_DEV, $INET_DEV } oifname $DMZ_DEV ip daddr 198.51.100.5 tcp dport
443 accept

    # IPv6 access from internet to the HTTPS server in the DMZ
    iifname $INET_DEV oifname $DMZ_DEV ip6 daddr 2001:db8:b::5 tcp dport 443 accept

    # Access from LAN and DMZ to HTTPS servers on the internet
    iifname { $LAN_DEV, $DMZ_DEV } oifname $INET_DEV tcp dport 443 accept

```

```

# Last action: Log blocked packets
log prefix "nft drop FWD: "
}

# Postrouting chain to handle SNAT
chain postrouting {
    type nat hook postrouting priority srcnat; policy accept;

    # SNAT for IPv4 traffic from LAN to internet
    iifname $LAN_DEV oifname $INET_DEV snat ip to 203.0.113.1
}
}

```

2. Include the `/etc/nftables/firewall.nft` script in the `/etc/sysconfig/nftables.conf` file:

```
include "/etc/nftables/firewall.nft"
```

3. Enable IPv4 forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

4. Enable and start the `nftables` service:

```
# systemctl enable --now nftables
```

Verification

1. Optional: Verify the `nftables` rule set:

```
# nft list ruleset
...
```

2. Try to perform an access that the firewall prevents. For example, try to access the router using SSH from the DMZ:

```
# ssh router.example.com
ssh: connect to host router.example.com port 22: Network is unreachable
```

3. Depending on your logging settings, search:

- The `systemd` journal for the blocked packets:

```
# journalctl -k -g "nft drop"
Oct 14 17:27:18 router kernel: nft drop IN : IN=enp8s0 OUT= MAC=...
SRC=198.51.100.5 DST=198.51.100.1 ... PROTO=TCP SPT=40464 DPT=22 ... SYN ...
```

- The `/var/log/nftables.log` file for the blocked packets:

```
Oct 14 17:27:18 router kernel: nft drop IN : IN=enp8s0 OUT= MAC=...
SRC=198.51.100.5 DST=198.51.100.1 ... PROTO=TCP SPT=40464 DPT=22 ... SYN ...
```


2.8. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS

You can use **nftables** to limit the number of connections or to block IP addresses that attempt to establish a given amount of connections to prevent them from using too many system resources.

2.8.1. Limiting the number of connections by using nftables

By using the **ct count** parameter of the **nft** utility, you can limit the number of simultaneous connections per IP address. For example, you can use this feature to configure that each source IP address can only establish two parallel SSH connections to a host.

Procedure

1. Create the **filter** table with the **inet** address family:

```
# nft add table inet filter
```

2. Add the **input** chain to the **inet filter** table:

```
# nft add chain inet filter input { type filter hook input priority 0 \; }
```

3. Create a dynamic set for IPv4 addresses:

```
# nft add set inet filter limit-ssh { type ipv4_addr\; flags dynamic \; }
```

4. Add a rule to the **input** chain that allows only two simultaneous incoming connections to the SSH port (22) from an IPv4 address and rejects all further connections from the same IP:

```
# nft add rule inet filter input tcp dport ssh ct state new add @limit-ssh { ip saddr ct count over 2 } counter reject
```

Verification

1. Establish more than two new simultaneous SSH connections from the same IP address to the host. Nftables refuses connections to the SSH port if two connections are already established.
2. Display the **limit-ssh** dynamic set:

```
# nft list set inet filter limit-ssh
table inet filter {
  set limit-ssh {
    type ipv4_addr
    size 65535
    flags dynamic
    elements = { 192.0.2.1 ct count over 2 , 192.0.2.2 ct count over 2 }
  }
}
```

The **elements** entry displays addresses that currently match the rule. In this example, **elements** lists IP addresses that have active connections to the SSH port. Note that the output does not display the number of active connections or if connections were rejected.

2.8.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute

You can temporarily block hosts that are establishing more than ten IPv4 TCP connections within one minute.

Procedure

1. Create the **filter** table with the **ip** address family:

```
# nft add table ip filter
```

2. Add the **input** chain to the **filter** table:

```
# nft add chain ip filter input { type filter hook input priority 0 \; }
```

3. Add a set named **denylist** to the **filter** table:

```
# nft add set ip filter denylist { type ipv4_addr \; flags dynamic, timeout \; timeout 5m \; }
```

This command creates a dynamic set for IPv4 addresses. The **timeout 5m** parameter defines that **nftables** automatically removes entries after five minutes to prevent that the set fills up with stale entries.

4. Add a rule that automatically adds the source IP address of hosts that attempt to establish more than ten new TCP connections within one minute to the **denylist** set:

```
# nft add rule ip filter input ip protocol tcp ct state new, untracked add @denylist { ip saddr limit rate over 10/minute } drop
```

Additional resources

- [Using named sets in nftables](#)

2.9. DEBUGGING NFTABLES RULES

The **nftables** framework provides different options for administrators to debug rules and if packets match them.

2.9.1. Creating a rule with a counter

To identify if a rule is matched, you can use a counter.

- For more information about a procedure that adds a counter to an existing rule, see [Adding a counter to an existing rule](#).

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. Add a new rule with the **counter** parameter to the chain. The following example adds a rule with a counter that allows TCP traffic on port 22 and counts the packets and traffic that match this rule:

```
# nft add rule inet example_table example_chain tcp dport 22 counter accept
```

2. To display the counter values:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

2.9.2. Adding a counter to an existing rule

To identify if a rule is matched, you can use a counter.

- For more information about a procedure that adds a new rule with a counter, see [Creating a rule with the counter](#).

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the counter by replacing the rule but with the **counter** parameter. The following example replaces the rule displayed in the previous step and adds a counter:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 counter accept
```

3. To display the counter values:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

■

2.9.3. Monitoring packets that match an existing rule

The tracing feature in **nftables** in combination with the **nft monitor** command enables administrators to display packets that match a rule. You can enable tracing for a rule and use it to monitor packets that match this rule.

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the tracing feature by replacing the rule but with the **meta nftrace set 1** parameters. The following example replaces the rule displayed in the previous step and enables tracing:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 meta nftrace
set 1 accept
```

3. Use the **nft monitor** command to display the tracing. The following example filters the output of the command to display only entries that contain **inet example_table example_chain**:

```
# nft monitor | grep "inet example_table example_chain"
trace id 3c5eb15e inet example_table example_chain packet: iif "enp1s0" ether saddr
52:54:00:17:ff:e4 ether daddr 52:54:00:72:2f:6e ip saddr 192.0.2.1 ip daddr 192.0.2.2 ip dscp
cs0 ip ecn not-ect ip ttl 64 ip id 49710 ip protocol tcp ip length 60 tcp sport 56728 tcp dport
ssh tcp flags == syn tcp window 64240
trace id 3c5eb15e inet example_table example_chain rule tcp dport ssh nftrace set 1 accept
(verdict accept)
...
```



WARNING

Depending on the number of rules with tracing enabled and the amount of matching traffic, the **nft monitor** command can display a lot of output. Use **grep** or other utilities to filter the output.

2.10. BACKING UP AND RESTORING THE NFTABLES RULE SET

You can backup **nftables** rules to a file and later restoring them. Also, administrators can use a file with the rules to, for example, transfer the rules to a different server.

2.10.1. Backing up the nftables rule set to a file

You can use the **nft** utility to back up the **nftables** rule set to a file.

Procedure

- To backup **nftables** rules:
 - In a format produced by **nft list ruleset** format:

```
# nft list ruleset > file.nft
```

- In JSON format:

```
# nft -j list ruleset > file.json
```

2.10.2. Restoring the nftables rule set from a file

You can restore the **nftables** rule set from a file.

Procedure

- To restore **nftables** rules:
 - If the file to restore is in the format produced by **nft list ruleset** or contains **nft** commands directly:

```
# nft -f file.nft
```

- If the file to restore is in JSON format:

```
# nft -j -f file.json
```

2.11. ADDITIONAL RESOURCES

- [Using nftables in Red Hat Enterprise Linux 8](#)
- [What comes after iptables? Its successor, of course: nftables](#)
- [Firewalld: The Future is nftables](#)

CHAPTER 3. USING XDP-FILTER FOR HIGH-PERFORMANCE TRAFFIC FILTERING TO PREVENT DDOS ATTACKS

Compared to packet filters, such as **nftables**, Express Data Path (XDP) processes and drops network packets right at the network interface. Therefore, XDP determines the next step for the package before it reaches a firewall or other applications. As a result, XDP filters require less resources and can process network packets at a much higher rate than conventional packet filters to defend against distributed denial of service (DDoS) attacks. For example, during testing, Red Hat dropped 26 million network packets per second on a single core, which is significantly higher than the drop rate of **nftables** on the same hardware.

The **xdp-filter** utility allows or drops incoming network packets using XDP. You can create rules to filter traffic to or from specific:

- IP addresses
- MAC addresses
- Ports

Note that, even if **xdp-filter** has a significantly higher packet-processing rate, it does not have the same capabilities as, for example, **nftables**. Consider **xdp-filter** a conceptual utility to demonstrate packet filtering using XDP. Additionally, you can use the code of the utility for a better understanding of how to write your own XDP applications.



IMPORTANT

On other architectures than AMD and Intel 64-bit, the **xdp-filter** utility is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See [Technology Preview Features Support Scope](#) on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

3.1. DROPPING NETWORK PACKETS THAT MATCH AN XDP-FILTER RULE

You can use **xdp-filter** to drop network packets:

- To a specific destination port
- From a specific IP address
- From a specific MAC address

The **allow** policy of **xdp-filter** defines that all traffic is allowed and the filter drops only network packets that match a particular rule. For example, use this method if you know the source IP addresses of packets you want to drop.

Prerequisites

- The **xdp-tools** package is installed.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process incoming packets on a certain interface, such as **enp1s0**:

```
# xdp-filter load enp1s0
```

By default, **xdp-filter** uses the **allow** policy, and the utility drops only traffic that matches any rule.

Optionally, use the **-f feature** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of packet processing. To enable multiple features, separate them with a comma.

If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to drop packets that match them. For example:

- To drop incoming packets to port **22**, enter:

```
# xdp-filter port 22
```

This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, use the **-p protocol** option.

- To drop incoming packets from **192.0.2.1**, enter:

```
# xdp-filter ip 192.0.2.1 -m src
```

Note that **xdp-filter** does not support IP ranges.

- To drop incoming packets from MAC address **00:53:00:AA:07:BE**, enter:

```
# xdp-filter ether 00:53:00:AA:07:BE -m src
```

Verification

- Use the following command to display statistics about dropped and allowed packets:

```
# xdp-filter status
```

Additional resources

- **xdp-filter(8)** man page on your system
- If you are a developer and interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

3.2. DROPPING ALL NETWORK PACKETS EXCEPT THE ONES THAT MATCH AN XDP-FILTER RULE

You can use **xdp-filter** to allow only network packets:

- From and to a specific destination port
- From and to a specific IP address
- From and to specific MAC address

To do so, use the **deny** policy of **xdp-filter** which defines that the filter drops all network packets except the ones that match a particular rule. For example, use this method if you do not know the source IP addresses of packets you want to drop.



WARNING

If you set the default policy to **deny** when you load **xdp-filter** on an interface, the kernel immediately drops all packets from this interface until you create rules that allow certain traffic. To avoid being locked out from the system, enter the commands locally or connect through a different network interface to the host.

Prerequisites

- The **xdp-tools** package is installed.
- You are logged in to the host either locally or using a network interface for which you do not plan to filter the traffic.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process packets on a certain interface, such as **enp1s0**:

```
# xdp-filter load enp1s0 -p deny
```

Optionally, use the **-f feature** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of packet processing. To enable multiple features, separate them with a comma.

If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to allow packets that match them. For example:

- To allow packets to port **22**, enter:

```
# xdp-filter port 22
```

This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, pass the **-p protocol** option to the command.

- To allow packets to **192.0.2.1**, enter:


```
# xdp-filter ip 192.0.2.1
```

Note that **xdp-filter** does not support IP ranges.

- To allow packets to MAC address **00:53:00:AA:07:BE**, enter:

```
# xdp-filter ether 00:53:00:AA:07:BE
```



IMPORTANT

The **xdp-filter** utility does not support stateful packet inspection. This requires that you either do not set a mode using the **-m mode** option or you add explicit rules to allow incoming traffic that the machine receives in reply to outgoing traffic.

Verification

- Use the following command to display statistics about dropped and allowed packets:

```
# xdp-filter status
```

Additional resources

- **xdp-filter(8)** man page on your system
- If you are a developer and you are interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.