Red Hat Enterprise Linux 10

Managing networking infrastructure services

A guide to managing networking infrastructure services

Last Updated: 2025-11-26

Red Hat Enterprise Linux 10 Managing networking infrastructure services

A guide to managing networking infrastructure services

Legal Notice

Abstract

This document describes how to set up and manage networking core infrastructure services, such as DNS and DHCP, on Red Hat Enterprise Linux.

Table of Contents

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. SETTING UP AND CONFIGURING A BIND DNS SERVER

BIND is a feature-rich DNS server that is fully compliant with the Internet Engineering Task Force (IETF) DNS standards and draft standards. For example, administrators frequently use BIND as:

- Caching DNS server in the local network
- Authoritative DNS server for zones
- Secondary server to provide high availability for zones

1.1. CONSIDERATIONS ABOUT PROTECTING BIND WITH SELINUX OR RUNNING IT IN A CHANGE-ROOT ENVIRONMENT

To secure a BIND installation, you can:

- Run the **named** service without a change-root environment. In this case, SELinux in **enforcing** mode prevents exploitation of known BIND security vulnerabilities. By default, Red Hat Enterprise Linux uses SELinux in **enforcing** mode.

  IMPORTANT

  Running BIND on RHEL with SELinux in **enforcing** mode is more secure than running BIND in a change-root environment.

- Run the **named-chroot** service in a change-root environment.
  By using the change-root feature, administrators can define that the root directory of a process and its sub-processes is different to the / directory. When you start the **named-chroot** service, BIND switches its root directory to **/var/named/chroot/**. As a consequence, the service uses **mount --bind** commands to make the files and directories listed in **/etc/named-chroot.files** available in **/var/named/chroot/**, and the process has no access to files outside of **/var/named/chroot/**.

If you decide to use BIND:

- In normal mode, use the **named** service.
- In a change-root environment, use the **named-chroot** service. This requires that you install, additionally, the **named-chroot** package.

1.2. CONFIGURING BIND AS A CACHING DNS SERVER

By default, the BIND DNS server resolves and caches successful and failed lookups. The service then answers requests to the same records from its cache. This significantly improves the speed of DNS lookups.

Prerequisites

- The IP address of the server is static.

Procedure

1. Install the **bind** and **bind-utils** packages:

```
# dnf install bind bind-utils
```

2. If you want to run BIND in a change-root environment install the **bind-chroot** package:

```
# dnf install bind-chroot
```

   Note that running BIND on a host with SELinux in **enforcing** mode, which is default, is more secure.

3. Edit the **/etc/named.conf** file, and make the following changes in the **options** statement:

   a. Update the **listen-on** and **listen-on-v6** statements to specify on which IPv4 and IPv6 interfaces BIND should listen:

   ```
   listen-on port 53 { 127.0.0.1; 192.0.2.1; };
   listen-on-v6 port 53 { ::1; 2001:db8:1::1; };
   ```

   b. Update the **allow-query** statement to configure from which IP addresses and ranges clients can query this DNS server:

   ```
   allow-query { localhost; 192.0.2.0/24; 2001:db8:1::/64; };
   ```

   c. Add an **allow-recursion** statement to define from which IP addresses and ranges BIND accepts recursive queries:

   ```
   allow-recursion { localhost; 192.0.2.0/24; 2001:db8:1::/64; };
   ```

   WARNING

   Do not allow recursion on public IP addresses of the server. Otherwise, the server can become part of large-scale DNS amplification attacks.

   d. By default, BIND resolves queries by recursively querying from the root servers to an authoritative DNS server. Alternatively, you can configure BIND to forward queries to other DNS servers, such as the ones of your provider. In this case, add a **forwarders** statement with the list of IP addresses of the DNS servers that BIND should forward queries to:

   ```
   forwarders { 198.51.100.1; 203.0.113.5; };
   ```

   As a fall-back behavior, BIND resolves queries recursively if the forwarder servers do not respond. To disable this behavior, add a **forward only;** statement.

4. Verify the syntax of the **/etc/named.conf** file:

```
# named-checkconf
```

If the command displays no output, the syntax is correct.

5. Update the **firewalld** rules to allow incoming DNS traffic:

```
# firewall-cmd --permanent --add-service=dns
# firewall-cmd --reload
```

6. Start and enable BIND:

```
# systemctl enable --now named
```

If you want to run BIND in a change-root environment, use the **systemctl enable --now named-chroot** command to enable and start the service.

Verification

1. Use the newly set up DNS server to resolve a domain:

```
# dig @localhost www.example.org
...
www.example.org.    86400    IN    A    198.51.100.34

;; Query time: 917 msec
...
```

This example assumes that BIND runs on the same host and responds to queries on the **localhost** interface.

After querying a record for the first time, BIND adds the entry to its cache.

2. Repeat the previous query:

```
# dig @localhost www.example.org
...
www.example.org.    85332    IN    A    198.51.100.34

;; Query time: 1 msec
...
```

Because of the cached entry, further requests for the same record are significantly faster until the entry expires.

Additional resources

- **named.conf(5)** man page on your system
- **/usr/share/doc/bind/sample/etc/named.conf**

1.3. CONFIGURING LOGGING ON A BIND DNS SERVER

The configuration in the default **/etc/named.conf** file, as provided by the **bind** package, uses the **default_debug** channel and logs messages to the **/var/named/data/named.run** file. The **default_debug** channel only logs entries when the server's debug level is non-zero.

By using, different channels and categories, you can configure BIND to write different events with a defined severity to separate files.

Prerequisites

- BIND is already configured, for example, as a caching name server.
- The **named** or **named-chroot** service is running.

Procedure

1. Edit the **/etc/named.conf** file, and add **category** and **channel** phrases to the **logging** statement, for example:

```
logging {
  ...

  category notify { zone_transfer_log; };
  category xfer-in { zone_transfer_log; };
  category xfer-out { zone_transfer_log; };
  channel zone_transfer_log {
    file "/var/named/log/transfer.log" versions 10 size 50m;
    print-time yes;
    print-category yes;
    print-severity yes;
    severity info;
   };

  ...
};
```

   With this example configuration, BIND logs messages related to zone transfers to **/var/named/log/transfer.log**. BIND creates up to **10** versions of the log file and rotates them if they reach a maximum size of **50** MB.

   The **category** phrase defines to which channels BIND sends messages of a category.

   The **channel** phrase defines the destination of log messages including the number of versions, the maximum file size, and the severity level BIND should log to a channel. Additional settings, such as enabling logging the time stamp, category, and severity of an event are optional, but useful for debugging purposes.

2. Create the log directory if it does not exist, and grant write permissions to the **named** user on this directory:

```
# mkdir /var/named/log/
# chown named:named /var/named/log/
# chmod 700 /var/named/log/
```

3. Verify the syntax of the **/etc/named.conf** file:

```
# named-checkconf
```

   If the command displays no output, the syntax is correct.

4. Restart BIND:

```
# systemctl restart named
```

If you run BIND in a change-root environment, use the **systemctl restart named-chroot** command to restart the service.

Verification

- Display the content of the log file:

```
# cat /var/named/log/transfer.log
...
06-Jul-2022 15:08:51.261 xfer-out: info: client @0x7fecbc0b0700 192.0.2.2#36121/key example-transfer-key (example.com): transfer of 'example.com/IN': AXFR started: TSIG example-transfer-key (serial 2022070603)
06-Jul-2022 15:08:51.261 xfer-out: info: client @0x7fecbc0b0700 192.0.2.2#36121/key example-transfer-key (example.com): transfer of 'example.com/IN': AXFR ended
```

Additional resources

- **named.conf(5)** man page on your system

1.4. WRITING BIND ACLS

Controlling access to certain features of BIND can prevent unauthorized access and attacks, such as denial of service (DoS). BIND access control list (**acl**) statements are lists of IP addresses and ranges. Each ACL has a nickname that you can use in several statements, such as **allow-query**, to refer to the specified IP addresses and ranges.

> **WARNING**
>
> BIND uses only the first matching entry in an ACL. For example, if you define an ACL **{ 192.0.2/24; !192.0.2.1; }** and the host with IP address **192.0.2.1** connects, access is granted even if the second entry excludes this address.

BIND has the following built-in ACLs:

- **none**: Matches no hosts.
- **any**: Matches all hosts.
- **localhost**: Matches the loopback addresses **127.0.0.1** and **::1**, as well as the IP addresses of all interfaces on the server that runs BIND.
- **localnets**: Matches the loopback addresses **127.0.0.1** and **::1**, as well as all subnets the server that runs BIND is directly connected to.

Prerequisites

- BIND is already configured, for example, as a caching name server.
- The **named** or **named-chroot** service is running.

Procedure

1. Edit the **/etc/named.conf** file and make the following changes:

    a. Add **acl** statements to the file. For example, to create an ACL named **internal-networks** for **127.0.0.1**, **192.0.2.0/24**, and **2001:db8:1::/64**, enter:

    ```
    acl internal-networks { 127.0.0.1; 192.0.2.0/24; 2001:db8:1::/64; };
    acl dmz-networks { 198.51.100.0/24; 2001:db8:2::/64; };
    ```

    b. Use the ACL's nickname in statements that support them, for example:

    ```
    allow-query { internal-networks; dmz-networks; };
    allow-recursion { internal-networks; };
    ```

2. Verify the syntax of the **/etc/named.conf** file:

    ```
    # named-checkconf
    ```

    If the command displays no output, the syntax is correct.

3. Reload BIND:

    ```
    # systemctl reload named
    ```

    If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

Verification

- Execute an action that triggers a feature which uses the configured ACL. For example, the ACL in this procedure allows only recursive queries from the defined IP addresses. In this case, enter the following command on a host that is not within the ACL's definition to attempt resolving an external domain:

    ```
    # dig +short @192.0.2.1 www.example.com
    ```

    If the command returns no output, BIND denied access, and the ACL works. For a verbose output on the client, use the command without **+short** option:

    ```
    # dig @192.0.2.1 www.example.com
    ...
    ;; WARNING: recursion requested but not available
    ...
    ```

1.5. CONFIGURING ZONES ON A BIND DNS SERVER

A DNS zone is a database with resource records for a specific sub-tree in the domain space. For example, if you are responsible for the **example.com** domain, you can set up a zone for it in BIND. As a result, clients can, resolve **www.example.com** to the IP address configured in this zone.

1.5.1. The SOA record in zone files

The start of authority (SOA) record is a required record in a DNS zone. This record is important, for example, if multiple DNS servers are authoritative for a zone but also to DNS resolvers.

A SOA record in BIND has the following syntax:

```
name class type mname rname serial refresh retry expire minimum
```

For better readability, administrators typically split the record in zone files into multiple lines with comments that start with a semicolon (**;**). Note that, if you split a SOA record, parentheses keep the record together:

```
@ IN SOA ns1.example.com. hostmaster.example.com. (
                          2022070601 ; serial number
                          1d         ; refresh period
                          3h         ; retry period
                          3d         ; expire time
                          3h )       ; minimum TTL
```

IMPORTANT

Note the trailing dot at the end of the fully-qualified domain names (FQDNs). FQDNs consist of multiple domain labels, separated by dots. Because the DNS root has an empty label, FQDNs end with a dot. Therefore, BIND appends the zone name to names without a trailing dot. A hostname without a trailing dot, for example, **ns1.example.com** would be expanded to **ns1.example.com.example.com.**, which is not the correct address of the primary name server.

These are the fields in a SOA record:

- **name**: The name of the zone, the so-called **origin**. If you set this field to **@**, BIND expands it to the zone name defined in **/etc/named.conf**.
- **class**: In SOA records, you must set this field always to Internet (**IN**).
- **type**: In SOA records, you must set this field always to **SOA**.
- **mname** (master name): The hostname of the primary name server of this zone.
- **rname** (responsible name): The email address of who is responsible for this zone. Note that the format is different. You must replace the at sign (**@**) with a dot (**.**).
- **serial**: The version number of this zone file. Secondary name servers only update their copies of the zone if the serial number on the primary server is higher.
  The format can be any numeric value. A commonly-used format is **<year><month><day><two-digit-number>**. With this format, you can, theoretically, change the zone file up to a hundred times per day.

- **refresh**: The amount of time secondary servers should wait before checking the primary server if the zone was updated.
- **retry**: The amount of time after that a secondary server retries to query the primary server after a failed attempt.
- **expire**: The amount of time after that a secondary server stops querying the primary server, if all previous attempts failed.
- **minimum**: RFC 2308 changed the meaning of this field to the negative caching time. Compliant resolvers use it to determine how long to cache **NXDOMAIN** name errors.

> NOTE
>
> A numeric value in the **refresh**, **retry**, **expire**, and **minimum** fields define a time in seconds. However, for better readability, use time suffixes, such as **m** for minute, **h** for hours, and **d** for days. For example, **3h** stands for 3 hours.

1.5.2. Setting up a forward zone on a BIND primary server

Forward zones map names to IP addresses and other information. For example, if you are responsible for the domain **example.com**, you can set up a forward zone in BIND to resolve names, such as **www.example.com**.

Prerequisites

- BIND is already configured, for example, as a caching name server.
- The **named** or **named-chroot** service is running.

Procedure

1. Add a zone definition to the **/etc/named.conf** file:

```
zone "example.com" {
   type master;
   file "example.com.zone";
   allow-query { any; };
   allow-transfer { none; };
};
```

   These settings define:

   - This server as the primary server (**type master**) for the **example.com** zone.
   - The **/var/named/example.com.zone** file is the zone file. If you set a relative path, as in this example, this path is relative to the directory you set in **directory** in the **options** statement.
   - Any host can query this zone. Alternatively, specify IP ranges or BIND access control list (ACL) nicknames to limit the access.
   - No host can transfer the zone. Allow zone transfers only when you set up secondary servers and only for the IP addresses of the secondary servers.

2. Verify the syntax of the **/etc/named.conf** file:

```
# named-checkconf
```

If the command displays no output, the syntax is correct.

3. Create the **/var/named/example.com.zone** file, for example, with the following content:

```
$TTL 8h
@ IN SOA ns1.example.com. hostmaster.example.com. (
                          2022070601 ; serial number
                          1d         ; refresh period
                          3h         ; retry period
                          3d         ; expire time
                          3h )       ; minimum TTL

                  IN NS   ns1.example.com.
                  IN MX   10 mail.example.com.

www               IN A    192.0.2.30
www               IN AAAA 2001:db8:1::30
ns1               IN A    192.0.2.1
ns1               IN AAAA 2001:db8:1::1
mail              IN A    192.0.2.20
mail              IN AAAA 2001:db8:1::20
```

This zone file:

- Sets the default time-to-live (TTL) value for resource records to 8 hours. Without a time suffix, such as **h** for hour, BIND interprets the value as seconds.
- Contains the required SOA resource record with details about the zone.
- Sets **ns1.example.com** as an authoritative DNS server for this zone. To be functional, a zone requires at least one name server (**NS**) record. However, to be compliant with RFC 1912, you require at least two name servers.
- Sets **mail.example.com** as the mail exchanger (**MX**) of the **example.com** domain. The numeric value in front of the host name is the priority of the record. Entries with a lower value have a higher priority.
- Sets the IPv4 and IPv6 addresses of **www.example.com**, **mail.example.com**, and **ns1.example.com**.

4. Set secure permissions on the zone file that allow only the **named** group to read it:

```
# chown root:named /var/named/example.com.zone
# chmod 640 /var/named/example.com.zone
```

5. Verify the syntax of the **/var/named/example.com.zone** file:

```
# named-checkzone example.com /var/named/example.com.zone
zone example.com/IN: loaded serial 2022070601
OK
```

6. Reload BIND:

```
# systemctl reload named
```

If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

Verification

- Query different records from the **example.com** zone, and verify that the output matches the records you have configured in the zone file:

```
# dig +short @localhost AAAA www.example.com
2001:db8:1::30

# dig +short @localhost NS example.com
ns1.example.com.

# dig +short @localhost A ns1.example.com
192.0.2.1
```

This example assumes that BIND runs on the same host and responds to queries on the **localhost** interface.

Additional resources

- The SOA record in zone files
- Writing BIND ACLs
- RFC 1912 - Common DNS operational and configuration errors

1.5.3. Setting up a reverse zone on a BIND primary server

Reverse zones map IP addresses to names. For example, if you are responsible for IP range **192.0.2.0/24**, you can set up a reverse zone in BIND to resolve IP addresses from this range to hostnames.

> **NOTE**
>
> If you create a reverse zone for whole classful networks, name the zone accordingly. For example, for the class C network **192.0.2.0/24**, the name of the zone is **2.0.192.in-addr.arpa**. If you want to create a reverse zone for a different network size, for example **192.0.2.0/28**, the name of the zone is **28-2.0.192.in-addr.arpa**.

Prerequisites

- BIND is already configured, for example, as a caching name server.
- The **named** or **named-chroot** service is running.

Procedure

1. Add a zone definition to the **/etc/named.conf** file:

```
zone "2.0.192.in-addr.arpa" {
  type master;
  file "2.0.192.in-addr.arpa.zone";
  allow-query { any; };
  allow-transfer { none; };
};
```

These settings define:

- This server as the primary server (**type master**) for the **2.0.192.in-addr.arpa** reverse zone.
- The **/var/named/2.0.192.in-addr.arpa.zone** file is the zone file. If you set a relative path, as in this example, this path is relative to the directory you set in **directory** in the **options** statement.
- Any host can query this zone. Alternatively, specify IP ranges or BIND access control list (ACL) nicknames to limit the access.
- No host can transfer the zone. Allow zone transfers only when you set up secondary servers and only for the IP addresses of the secondary servers.

2. Verify the syntax of the **/etc/named.conf** file:

```
# named-checkconf
```

If the command displays no output, the syntax is correct.

3. Create the **/var/named/2.0.192.in-addr.arpa.zone** file, for example, with the following content:

```
$TTL 8h
@ IN SOA ns1.example.com. hostmaster.example.com. (
                    2022070601 ; serial number
                    1d         ; refresh period
                    3h         ; retry period
                    3d         ; expire time
                    3h )       ; minimum TTL

              IN NS   ns1.example.com.

1             IN PTR  ns1.example.com.
30            IN PTR  www.example.com.
```

This zone file:

- Sets the default time-to-live (TTL) value for resource records to 8 hours. Without a time suffix, such as **h** for hour, BIND interprets the value as seconds.
- Contains the required SOA resource record with details about the zone.
- Sets **ns1.example.com** as an authoritative DNS server for this reverse zone. To be functional, a zone requires at least one name server (**NS**) record. However, to be compliant with RFC 1912, you require at least two name servers.
- Sets the pointer (**PTR**) record for the **192.0.2.1** and **192.0.2.30** addresses.

4. Set secure permissions on the zone file that only allow the **named** group to read it:

```
# chown root:named /var/named/2.0.192.in-addr.arpa.zone
# chmod 640 /var/named/2.0.192.in-addr.arpa.zone
```

5. Verify the syntax of the **/var/named/2.0.192.in-addr.arpa.zone** file:

```
# named-checkzone 2.0.192.in-addr.arpa /var/named/2.0.192.in-addr.arpa.zone
zone 2.0.192.in-addr.arpa/IN: loaded serial 2022070601
OK
```

6. Reload BIND:

```
# systemctl reload named
```

If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

Verification

- Query different records from the reverse zone, and verify that the output matches the records you have configured in the zone file:

```
# dig +short @localhost -x 192.0.2.1
ns1.example.com.

# dig +short @localhost -x 192.0.2.30
www.example.com.
```

This example assumes that BIND runs on the same host and responds to queries on the **localhost** interface.

Additional resources

- The SOA record in zone files
- Writing BIND ACLs
- RFC 1912 - Common DNS operational and configuration errors

1.5.4. Updating a BIND zone file

In certain situations, for example if an IP address of a server changes, you must update a zone file. If multiple DNS servers are responsible for a zone, perform this procedure only on the primary server. Other DNS servers that store a copy of the zone will receive the update through a zone transfer.

Prerequisites

- The zone is configured.
- The **named** or **named-chroot** service is running.

Procedure

1. Optional: Identify the path to the zone file in the **/etc/named.conf** file:

```
options {
    ...
    directory       "/var/named";
}

zone "example.com" {
    ...
    file "example.com.zone";
};
```

You find the path to the zone file in the **file** statement in the zone's definition. A relative path is relative to the directory set in **directory** in the **options** statement.

2. Edit the zone file:

   a. Make the required changes.

   b. Increment the serial number in the start of authority (SOA) record.

   > **IMPORTANT**
   >
   > If the serial number is equal to or lower than the previous value, secondary servers will not update their copy of the zone.

3. Verify the syntax of the zone file:

```
# named-checkzone example.com /var/named/example.com.zone
zone example.com/IN: loaded serial 2022062802
OK
```

4. Reload BIND:

```
# systemctl reload named
```

If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

Verification

- Query the record you have added, modified, or removed, for example:

```
# dig +short @localhost A ns2.example.com
192.0.2.2
```

This example assumes that BIND runs on the same host and responds to queries on the **localhost** interface.

Additional resources

- The SOA record in zone files
- Setting up a forward zone on a BIND primary server
- Setting up a reverse zone on a BIND primary server

1.5.5. DNSSEC zone signing using the automated key generation and zone maintenance features

You can sign zones with domain name system security extensions (DNSSEC) to ensure authentication and data integrity. Such zones contain additional resource records. Clients can use them to verify the authenticity of the zone information.

If you enable the DNSSEC policy feature for a zone, BIND performs the following actions automatically:

- Creates the keys
- Signs the zone
- Maintains the zone, including re-signing and periodically replacing the keys.

> **IMPORTANT**
>
> To enable external DNS servers to verify the authenticity of a zone, you must add the public key of the zone to the parent zone. Contact your domain provider or registry for further details on how to accomplish this.

This procedure uses the built-in **default** DNSSEC policy in BIND. This policy uses single **ECDSAP256SHA** key signatures. Alternatively, create your own policy to use custom keys, algorithms, and timings.

Prerequisites

- The **bind** package is installed.
- The zone for which you want to enable DNSSEC is configured.
- The **named** or **named-chroot** service is running.
- The server synchronizes the time with a time server. An accurate system time is important for DNSSEC validation.

Procedure

1. Edit the **/etc/named.conf** file, and add **dnssec-policy default;** and **inline-signing yes;** to the zone for which you want to enable DNSSEC:

```
zone "example.com" {
    ...
    dnssec-policy default;
    inline-signing yes;
};
```

2. Reload BIND:

```
# systemctl reload named
```

   If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

3. BIND stores the public key in the **/var/named/K*<zone_name>*.+*<algorithm>*+*<key_ID>*.key** file. Use this file to display the public key of the zone in the format that the parent zone requires:

   - DS record format:

```
# dnssec-dsfromkey /var/named/Kexample.com.+013+61141.key
example.com. IN DS 61141 13 2
3E184188CF6D2521EDFDC3F07CFEE8D0195AACBD85E68BAE0620F638B4B1B027
```

   - DNSKEY format:

```
# grep DNSKEY /var/named/Kexample.com.+013+61141.key
example.com. 3600 IN DNSKEY 257 3 13
sjzT3jNEp120aSO4mPEHHSkReHUf7AABNnT8hNRTzD5cKMQSjDJin2I3
5CaKVcWO1pm+HltxUEt+X9dfp8OZkg==
```

4. Request to add the public key of the zone to the parent zone. Contact your domain provider or registry for further details on how to accomplish this.

Verification

1. Query your own DNS server for a record from the zone for which you enabled DNSSEC signing:

```
# dig +dnssec +short @localhost A www.example.com
192.0.2.30
A 13 3 28800 20220718081258 20220705120353 61141 example.com.
e7Cfh6GuOBMAWsgsHSVTPh+JJSOI/Y6zctzIuqIU1JqEgOOAfL/Qz474
M0sgi54m1Kmnr2ANBKJN9uvOs5eXYw==
```

   This example assumes that BIND runs on the same host and responds to queries on the **localhost** interface.

2. After the public key has been added to the parent zone and propagated to other servers, verify that the server sets the authenticated data (**ad**) flag on queries to the signed zone:

```
#  dig @localhost example.com +dnssec
...
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
...
```

Additional resources

- Setting up a forward zone on a BIND primary server
- Setting up a reverse zone on a BIND primary server

1.6. CONFIGURING ZONE TRANSFERS AMONG BIND DNS SERVERS

Zone transfers ensure that all DNS servers that have a copy of the zone use up-to-date data.

Prerequisites

- On the future primary server, the zone for which you want to set up zone transfers is already configured.
- On the future secondary server, BIND is already configured, for example, as a caching name server.
- On both servers, the **named** or **named-chroot** service is running.

Procedure

1. On the existing primary server:

   a. Create a shared key, and append it to the **/etc/named.conf** file:

   ```
   # tsig-keygen example-transfer-key | tee -a /etc/named.conf
   key "example-transfer-key" {
           algorithm hmac-sha256;
           secret "q7ANbnyliDMuvWgnKOxMLi313JGcTZB5ydMW5CyUGXQ=";
   };
   ```

   This command displays the output of the **tsig-keygen** command and automatically appends it to **/etc/named.conf**.

   You will require the output of the command later on the secondary server as well.

   b. Edit the zone definition in the **/etc/named.conf** file:

      i. In the **allow-transfer** statement, define that servers must provide the key specified in the **example-transfer-key** statement to transfer a zone:

      ```
      zone "example.com" {
        ...
        allow-transfer { key example-transfer-key; };
      };
      ```

      Alternatively, use BIND access control list (ACL) nicknames in the **allow-transfer** statement.

      ii. By default, after a zone has been updated, BIND notifies all name servers which have a name server (**NS**) record in this zone. If you do not plan to add an **NS** record for the secondary server to the zone, you can, configure that BIND notifies this server anyway. For that, add the **also-notify** statement with the IP addresses of this secondary server to the zone:

      ```
      zone "example.com" {
        ...
        also-notify { 192.0.2.2; 2001:db8:1::2; };
      };
      ```

   c. Verify the syntax of the **/etc/named.conf** file:

   ```
   # named-checkconf
   ```

   If the command displays no output, the syntax is correct.

d. Reload BIND:

```
# systemctl reload named
```

If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

2. On the future secondary server:

a. Edit the **/etc/named.conf** file as follows:

i. Add the same key definition as on the primary server:

```
key "example-transfer-key" {
        algorithm hmac-sha256;
        secret "q7ANbnyliDMuvWgnKOxMLi313JGcTZB5ydMW5CyUGXQ=";
};
```

ii. Add the zone definition to the **/etc/named.conf** file:

```
zone "example.com" {
   type slave;
   file "slaves/example.com.zone";
   allow-query { any; };
   allow-transfer { none; };
   masters {
     192.0.2.1 key example-transfer-key;
     2001:db8:1::1 key example-transfer-key;
   };
};
```

These settings state:

- This server is a secondary server (**type slave**) for the **example.com** zone.
- The **/var/named/slaves/example.com.zone** file is the zone file. If you set a relative path, as in this example, this path is relative to the directory you set in **directory** in the **options** statement. To separate zone files for which this server is secondary from primary ones, you can store them, for example, in the **/var/named/slaves/** directory.
- Any host can query this zone. Alternatively, specify IP ranges or ACL nicknames to limit the access.
- No host can transfer the zone from this server.
- The IP addresses of the primary server of this zone are **192.0.2.1** and **2001:db8:1::2**. Alternatively, you can specify ACL nicknames. This secondary server will use the key named **example-transfer-key** to authenticate to the primary server.

b. Verify the syntax of the **/etc/named.conf** file:

```
# named-checkconf
```

c. Reload BIND:

```
# systemctl reload named
```

If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

3. Optional: Modify the zone file on the primary server and add an **NS** record for the new secondary server.

Verification

On the secondary server:

1. Display the **systemd** journal entries of the **named** service:

```
# journalctl -u named
...
Jul 06 15:08:51 ns2.example.com named[2024]: zone example.com/IN: Transfer started.
Jul 06 15:08:51 ns2.example.com named[2024]: transfer of 'example.com/IN' from
192.0.2.1#53: connected using 192.0.2.2#45803
Jul 06 15:08:51 ns2.example.com named[2024]: zone example.com/IN: transferred serial
2022070101
Jul 06 15:08:51 ns2.example.com named[2024]: transfer of 'example.com/IN' from
192.0.2.1#53: Transfer status: success
Jul 06 15:08:51 ns2.example.com named[2024]: transfer of 'example.com/IN' from
192.0.2.1#53: Transfer completed: 1 messages, 29 records, 2002 bytes, 0.003 secs (667333
bytes/sec)
```

If you run BIND in a change-root environment, use the **journalctl -u named-chroot** command to display the journal entries.

2. Verify that BIND created the zone file:

```
# ls -l /var/named/slaves/
total 4
-rw-r--r--. 1 named named 2736 Jul  6 15:08 example.com.zone
```

Note that, by default, secondary servers store zone files in a binary raw format.

3. Query a record of the transferred zone from the secondary server:

```
# dig +short @192.0.2.2 AAAA www.example.com
2001:db8:1::30
```

This example assumes that the secondary server you set up in this procedure listens on IP address **192.0.2.2**.

Additional resources

- Setting up a forward zone on a BIND primary server
- Setting up a reverse zone on a BIND primary server
- Writing BIND ACLs
- Updating a BIND zone file

1.7. CONFIGURING RESPONSE POLICY ZONES IN BIND TO OVERRIDE DNS RECORDS

By using DNS blocking and filtering, administrators can rewrite a DNS response to block access to certain domains or hosts. In BIND, response policy zones (RPZs) provide this feature. You can configure different actions for blocked entries, such as returning an **NXDOMAIN** error or not responding to the query.

If you have multiple DNS servers in your environment, use this procedure to configure the RPZ on the primary server, and later configure zone transfers to make the RPZ available on your secondary servers.

Prerequisites

- BIND is already configured, for example, as a caching name server.
- The **named** or **named-chroot** service is running.

Procedure

1. Edit the **/etc/named.conf** file, and make the following changes:

   a. Add a **response-policy** definition to the **options** statement:

   ```
   options {
     ...

     response-policy {
       zone "rpz.local";
     };

     ...
   }
   ```

   You can set a custom name for the RPZ in the **zone** statement in **response-policy**. However, you must use the same name in the zone definition in the next step.

   b. Add a **zone** definition for the RPZ you set in the previous step:

   ```
   zone "rpz.local" {
     type master;
     file "rpz.local";
     allow-query { localhost; 192.0.2.0/24; 2001:db8:1::/64; };
     allow-transfer { none; };
   };
   ```

   These settings state:

   - This server is the primary server (**type master**) for the RPZ named **rpz.local**.
   - The **/var/named/rpz.local** file is the zone file. If you set a relative path, as in this example, this path is relative to the directory you set in **directory** in the **options** statement.
   - Any hosts defined in **allow-query** can query this RPZ. Alternatively, specify IP ranges or BIND access control list (ACL) nicknames to limit the access.

- No host can transfer the zone. Allow zone transfers only when you set up secondary servers and only for the IP addresses of the secondary servers.

2. Verify the syntax of the **/etc/named.conf** file:

```
# named-checkconf
```

If the command displays no output, the syntax is correct.

3. Create the **/var/named/rpz.local** file, for example, with the following content:

```
$TTL 10m
@ IN SOA ns1.example.com. hostmaster.example.com. (
                2022070601 ; serial number
                1h         ; refresh period
                1m         ; retry period
                3d         ; expire time
                1m )       ; minimum TTL

          IN NS    ns1.example.com.

example.org      IN CNAME .
*.example.org    IN CNAME .
example.net      IN CNAME rpz-drop.
*.example.net    IN CNAME rpz-drop.
```

This zone file:

- Sets the default time-to-live (TTL) value for resource records to 10 minutes. Without a time suffix, such as **h** for hour, BIND interprets the value as seconds.
- Contains the required start of authority (SOA) resource record with details about the zone.
- Sets **ns1.example.com** as an authoritative DNS server for this zone. To be functional, a zone requires at least one name server (**NS**) record. However, to be compliant with RFC 1912, you require at least two name servers.
- Return an **NXDOMAIN** error for queries to **example.org** and hosts in this domain.
- Drop queries to **example.net** and hosts in this domain.

For a full list of actions and examples, see IETF draft: DNS Response Policy Zones (RPZ) .

4. Verify the syntax of the **/var/named/rpz.local** file:

```
# named-checkzone rpz.local /var/named/rpz.local
zone rpz.local/IN: loaded serial 2022070601
OK
```

5. Reload BIND:

```
# systemctl reload named
```

If you run BIND in a change-root environment, use the **systemctl reload named-chroot** command to reload the service.

Verification

1. Attempt to resolve a host in **example.org**, that is configured in the RPZ to return an **NXDOMAIN** error:

```
# dig @localhost www.example.org
...
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 30286
...
```

   This example assumes that BIND runs on the same host and responds to queries on the **localhost** interface.

2. Attempt to resolve a host in the **example.net** domain, that is configured in the RPZ to drop queries:

```
# dig @localhost www.example.net
...
;; connection timed out; no servers could be reached
...
```

Additional resources

- IETF draft: DNS Response Policy Zones (RPZ)

1.8. RECORDING DNS QUERIES BY USING DNSTAP

As a network administrator, you can record Domain Name System (DNS) details to analyze DNS traffic patterns, monitor DNS server performance, and troubleshoot DNS issues. If you want an advanced way to monitor and log details of incoming name queries, use the **dnstap** interface that records sent messages from the **named** service. You can capture and record DNS queries to collect information about websites or IP addresses.

Prerequisites

- The **bind** package is installed.

> **WARNING**
>
> If you already have a **BIND** version installed and running, adding a new version of **BIND** will overwrite the existing version.

Procedure

1. Enable **dnstap** and the target file by editing the **/etc/named.conf** file in the **options** block:

```
options
{
# ...
```

```
dnstap { all; }; # Configure filter
dnstap-output file "/var/named/data/dnstap.bin" versions 2;
# ...
};
# end of options
```

2. To specify which types of DNS traffic you want to log, add **dnstap** filters to the **dnstap** block in the **/etc/named.conf** file. You can use the following filters:
   - **auth** - Authoritative zone response or answer.
   - **client** - Internal client query or answer.
   - **forwarder** - Forwarded query or response from it.
   - **resolver** - Iterative resolution query or response.
   - **update** - Dynamic zone update requests.
   - **all** - Any from the above options.
   - **query** or **response** - If you do not specify a **query** or a **response** keyword, **dnstap** records both.

> NOTE
>
> The **dnstap** filter contains multiple definitions delimited by a **;** in the **dnstap {}** block with the following syntax: **dnstap { (all | auth | client | forwarder | resolver | update) [(query | response)]; … };**

3. To customize the behavior of the **dnstap** utility on the recorded packets, modify the **dnstap-output** option by providing additional parameters, as follows:
   - **size** (unlimited | <size>) - Enable automatic rolling over of the **dnstap** file when its size reaches the specified limit.
   - **versions** (unlimited | <integer>) - Specify the number of automatically rolled files to keep.
   - **suffix** (increment | timestamp) - Choose the naming convention for rolled out files. By default, the increment starts with **.0**. Alternatively, you can use the UNIX timestamp by setting the **timestamp** value.
     The following example requests **auth** responses only, **client** queries, and both queries and responses of dynamic **updates**:

     ```
     Example:

     dnstap {auth response; client query; update;};
     ```

4. To apply your changes, restart the **named** service:

   ```
   # systemctl restart named.service
   ```

5. Configure a periodic rollout for active logs

In the following example, the **cron** scheduler runs the content of the user-edited script once a day. The **roll** option with the value **3** specifies that **dnstap** can create up to three backup log files. The value **3** overrides the **version** parameter of the **dnstap-output** variable, and limits the number of backup log files to three. Additionally, the binary log file is moved to another directory and renamed, and it never reaches the **.2** suffix, even if three backup log files already exist. You can skip this step if automatic rolling of binary logs based on size limit is sufficient.

```
Example:

sudoedit /etc/cron.daily/dnstap

#!/bin/sh
rndc dnstap -roll 3
mv /var/named/data/dnstap.bin.1 /var/log/named/dnstap/dnstap-$(date -I).bin

# use dnstap-read to analyze saved logs
sudo chmod a+x /etc/cron.daily/dnstap
```

6. Handle and analyze logs in a human-readable format by using the **dnstap-read** utility:
   In the following example, the **dnstap-read** utility prints the output in the **YAML** file format.

```
Example:

dnstap-read  -p /var/named/data/dnstap.bin
```

CHAPTER 2. SETTING UP AN UNBOUND DNS SERVER

The **unbound** DNS server is a validating, recursive, and caching DNS resolver. Additionally, **unbound** focuses on security and has, for example, Domain Name System Security Extensions (DNSSEC) enabled by default.

2.1. CONFIGURING UNBOUND AS A CACHING DNS SERVER

By default, the **unbound** DNS service resolves and caches successful and failed lookups. The service then answers requests to the same records from its cache.

Procedure

1. Install the **unbound** package:

   ```
   # dnf install unbound
   ```

2. Edit the **/etc/unbound/unbound.conf** file, and make the following changes in the **server** clause:

   a. Add **interface** parameters to configure on which IP addresses the **unbound** service listens for queries, for example:

      ```
      interface: 127.0.0.1
      interface: 192.0.2.1
      interface: 2001:db8:1::1
      ```

      With these settings, **unbound** only listens on the specified IPv4 and IPv6 addresses.

      Limiting the interfaces to the required ones prevents clients from unauthorized networks, such as the internet, from sending queries to this DNS server.

   b. Add **access-control** parameters to configure from which subnets clients can query the DNS service, for example:

      ```
      access-control: 127.0.0.0/8 allow
      access-control: 192.0.2.0/24 allow
      access-control: 2001:db8:1::/64 allow
      ```

3. Create private keys and certificates for remotely managing the **unbound** service:

   ```
   # systemctl restart unbound-keygen
   ```

   If you skip this step, verifying the configuration in the next step will report the missing files. However, the **unbound** service automatically creates the files if they are missing.

4. Verify the configuration file:

   ```
   # unbound-checkconf
   unbound-checkconf: no errors in /etc/unbound/unbound.conf
   ```

5. Update the firewalld rules to allow incoming DNS traffic:

```
# firewall-cmd --permanent --add-service=dns
# firewall-cmd --reload
```

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

```
# systemctl enable --now unbound
```

Verification

1. Query the **unbound** DNS server listening on the **localhost** interface to resolve a domain:

```
# dig @localhost www.example.com
...
www.example.com.    86400    IN    A    198.51.100.34

;; Query time: 330 msec
...
```

After querying a record for the first time, **unbound** adds the entry to its cache.

2. Repeat the previous query:

```
# dig @localhost www.example.com
...
www.example.com.    85332    IN    A    198.51.100.34

;; Query time: 1 msec
...
```

Because of the cached entry, further requests for the same record are significantly faster until the entry expires.

Additional resources

- **unbound.conf(5)** man page on your system

CHAPTER 3. PROVIDING DHCP SERVICES

The dynamic host configuration protocol (DHCP) is a network protocol that automatically assigns IP information to clients. You can set up Kea to provide a DHCP server in your network.

3.1. THE DIFFERENCE BETWEEN STATIC AND DYNAMIC IP ADDRESSING

To manage how devices receive their unique network addresses, administrators can select between two primary methods: static and dynamic IP addressing.

Static IP addressing

When you assign a static IP address to a device, the address does not change over time unless you change it manually. Use static IP addressing if you want:

- To ensure network address consistency for servers such as DNS, and authentication servers.
- To use out-of-band management devices that work independently of other network infrastructure.

Dynamic IP addressing

When you configure a device to use a dynamic IP address, the address can change over time. For this reason, dynamic addresses are typically used for devices that connect to the network occasionally because the IP address can be different after rebooting the host.
Dynamic IP addresses are more flexible, easier to set up, and administer. DHCP is a traditional method of dynamically assigning network configurations to hosts.

NOTE

There is no strict rule defining when to use static or dynamic IP addresses. It depends on the user's needs, preferences, and the network environment.

3.2. DHCP TRANSACTION PHASES

DHCP works in four phases: Discovery, Offer, Request, Acknowledgment, also called the DORA process. DHCP uses this process to provide IP addresses to clients.

Discovery

The DHCP client sends a message to discover the DHCP server in the network. This message is broadcasted at the network and data link layer.

Offer

The DHCP server receives messages from the client and offers an IP address to the DHCP client. This message is unicast at the data link layer but broadcast at the network layer.

Request

The DHCP client requests the DHCP server for the offered IP address. This message is unicast at the data link layer but broadcast at the network layer.

Acknowledgment

The DHCP server sends an acknowledgment to the DHCP client. This message is unicast at the data link layer but broadcast at the network layer. It is the final message of the DHCP DORA process.

3.3. OVERVIEW OF USING KEA FOR DHCPV4 AND DHCPV6

A key aspect of Kea's design is that DHCPv4 and DHCPv6 are managed independently. Each protocol has its own configuration file and systemd service, giving you the flexibility to run a DHCPv4 server, a DHCPv6 server, or both to meet your network's requirements.

For each protocol, Kea uses a separate configuration file and service:

DHCPv4

- Configuration file: **/etc/kea/kea-dhcp4.conf**
- Systemd service name: **kea-dhcp4**

DHCPv6

- Configuration file: **/etc/kea/kea-dhcp6.conf**
- Systemd service name: **kea-dhcp6**

3.4. THE KEA LEASE DATABASE

A DHCP lease is the period for which Kea allocates a network address to a client. The lease databases contain information about the allocated leases, such as the IP address assigned to a media access control (MAC) address or the timestamp when the lease expires.

All timestamps in the lease databases are in Coordinated Universal Time (UTC).

Kea supports the following lease backends:

memfile (default)

A text-based file stored on the disk. By default, Kea stores the DHCP leases in the following files:

- For DHCPv4: **/var/lib/kea/kea-leases4.csv**
- For DHCPv6: **/var/lib/kea/kea-leases6.csv**

> **WARNING**
>
> Manually updating the files can cause inconsistencies and file corruption. For performance reasons, Kea stores the lease data in memory and does not monitor the lease files during runtime. Manual edits can be overridden the next time Kea updates the files.

mysql

A MySQL database backend.

pgsql

A PostgreSQL database backend.

For example, Kea updates the lease database in the following cases:

- On lease updates
- On graceful shutdown
- During periodic lease file cleanup (LFC) processes
- By requests through the API

3.5. SETTING UP A KEA DHCP SERVER

Kea is a modern, high-performance DHCP server with a modular design. Use a DHCP server to automatically assign IP addresses and other network settings to client devices. This eliminates the error-prone task of manual configuration.

Prerequisites

- The **kea** package is installed.
- You are logged in as the **root** user.

Procedure

1. If you are configuring an IPv4 network:

    a. Edit the **/etc/kea/kea-dhcp4.conf** file, and use the following configuration:

```
{
 "Dhcp4": {
   // Global settings that apply to all subnets unless overridden.
   "valid-lifetime": 86400,
   "option-data": [
     {
       "name": "domain-name",
 "data": "example.com"
     },
     {
       "name": "domain-name-servers",
 "data": "192.0.2.53"
     }
   ],

   // The network interfaces on which Kea will listen for DHCP traffic.
   "interfaces-config": {
     "interfaces": [ "enp1s0" ]
   },

   "subnet4": [
     // A definition of a subnet that is directly connected to the server
     {
       "id": 1,
       "subnet": "192.0.2.0/24",
       "pools": [
         { "pool": "192.0.2.20  - 192.0.2.100" },
```

```
        { "pool": "192.0.2.150 - 192.0.2.200" }
      ],
      "option-data": [
        { "name": "routers", "data": "192.0.2.1" }
      ],
    },

    // A definition of a remote subnet served through a DHCP relay
    {
      "id": 2,
      "subnet": "198.51.100.0/24",
      "pools": [
        { "pool": "198.51.100.20 - 198.51.100.100" }
      ],
  // Allowed DHCP relay agents
  "relay": {
        "ip-addresses": [ "198.51.100.5" ]
      },
      "option-data": [
        { "name": "routers", "data": "198.51.100.1" },
   { "name": "domain-name-servers", "data": "198.51.100.53" }
      ]
    }
  ]
 }
}
```

This example configures Kea to serve two subnets: one directly connected to the server and a remote one that uses a DHCP relay agent.

The settings specified in the example include the following:

interfaces

Defines the network interfaces on which Kea listens for DHCP requests. If a subnet is not directly connected to the server, ensure that you list the interface through which the subnet can be reached.

id

Defines a unique integer for the subnet. This is required if you define more than one subnet.

subnet

Defines the subnet in Classless Inter-Domain Routing (CIDR) format.

pools

Defines the IP address ranges from which Kea can assign addresses to clients.

option-data

Defines DHCP options sent to clients, such as the default gateway and DNS servers. Per-subnet option-data settings override global settings.

relay

Defines the IP addresses of DHCP relay agents. While this setting is optional for remote subnets, it improves the security to limit forwarded requests to trusted agents. Do not use this parameter for directly-connected subnets.

b. Verify the syntax of the configuration file:

```
# kea-dhcp4 -t /etc/kea/kea-dhcp4.conf
```

If the command returns **Syntax check failed**, fix the errors shown in the report.

c. Update the **firewalld** rules to allow incoming DHCPv4 traffic:

```
# firewall-cmd --permanent --add-service=dhcp
# firewall-cmd --reload
```

d. Enable and start the service:

```
# systemctl enable --now kea-dhcp4
```

2. If you are configuring an IPv6 network:

a. Edit the **/etc/kea/kea-dhcp6.conf** file, and use the following configuration:

```
{
 "Dhcp6": {
  // Global settings that apply to all subnets unless overridden.
  "valid-lifetime": 86400,
  "option-data": [
   {
    "name": "domain-name",
"data": "example.com"
   },
   {
    "name": "dns-servers",
"data": "2001:db8:0:1::53"
   }
  ],

  // The network interfaces on which Kea will listen for DHCP traffic.
  "interfaces-config": {
   "interfaces": [ "enp1s0" ]
  },

  "subnet6": [
   // A definition of a subnet that is directly connected to the server
   {
    "id": 1,
    "subnet": "2001:db8:0:1::/64",
    "pools": [
     { "pool": "2001:db8:0:1::1000 - 2001:db8:0:1::2000" },
     { "pool": "2001:db8:0:1::4000 - 2001:db8:0:1::5000" }
    ],
   },

   // A definition of a remote subnet served through a DHCP relay
   {
    "id": 2,
    "subnet": "2001:db8:0:2::/64",
    "pools": [
     { "pool": "2001:db8:0:2::1000 - 2001:db8:0:2::2000" }
    ],
```

```
// Allowed DHCP relay agents
"relay": {
        "ip-addresses": [ "2001:db8:0:2::5" ]
      },
      "option-data": [
  { "name": "dns-servers", "data": "2001:db8:0:1::53" }
      ]
    }
  ]
 }
}
```

This example configures Kea to serve two subnets: one directly connected to the server and a remote one that uses a DHCP relay agent.

b. Verify the syntax of the configuration file:

```
# kea-dhcp6 -t /etc/kea/kea-dhcp6.conf
```

If the command returns **Syntax check failed**, fix the errors shown in the report.

c. Update the **firewalld** rules to allow incoming DHCPv6 traffic:

```
# firewall-cmd --permanent --add-service=dhcpv6
# firewall-cmd --reload
```

d. Enable and start the service:

```
# systemctl enable --now kea-dhcp6
```

Verification

1. Configure a network connection with DHCP on a client. See Configuring an Ethernet connection by using **nmcli**.

2. Connect the client to the network.

3. Check if the client received an IP address from the DHCP server:

```
# ip address show <interface>
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:17:b8:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.20/24 brd 192.0.2.255 scope global noprefixroute enp1s0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1000/64 scope global noprefixroute
       valid_lft forever preferred_lft forever
```

Troubleshooting

- Check on which IPv4 and IPv6 addresses Kea is listening:

```
# ss -lunp | grep -E ':(67|547)'
```

If Kea does not listen on all interfaces you configured, check the **interfaces-config** setting in the Kea configuration files.

Next steps

- Configure logging

3.6. CONFIGURING LOGGERS IN KEA

If you want to customize log settings, such as the severity level, configure **loggers** in Kea.

By default, Kea writes log messages to:

- The systemd journal
- The **/var/log/messages** files, if the **rsyslogd** service is running

Prerequisites

- The **kea-dhcp4** and **kea-dhcp6** services are configured and running.
- You are logged in as the **root** user.

Procedure

1. If you are configuring an IPv4 network:

   a. Edit the **/etc/kea/kea-dhcp4.conf** file, and add the **loggers** configuration to the **Dhcp4** parameter. For example:

```
{
 "Dhcp4": {
   ...,
   "loggers":[
    {
     "name":"kea-dhcp4",
     "output-options":[
       {
        "output":"kea-dhcp4.log",
        "maxsize":104857600,
        "maxver":5
       }
     ],
     "severity":"INFO",
    }
   ],
   ...
```

   The settings specified in the example are:

   name

   Defines the name of the binary the **logger** settings apply to.

   output

Sets the log file name in the **/var/lib/kea/** directory.

maxsize

Sets the maximum size of the log file before Kea rotates it. The default value is **10240000** bytes.

maxver

Sets the maximum number of rotated versions Kea will keep. Note that a **maxsize** value less than **204800** bytes disables rotation.

severity

Specifies the category of messages logged. You can set one of the following values: **NONE**, **FATAL**, **ERROR**, **WARN**, **INFO**, and **DEBUG**. Kea logs only messages of the configured severity and above.

b. Verify the syntax of the configuration file:

```
# kea-dhcp4 -t /etc/kea/kea-dhcp4.conf
```

If the command returns **Syntax check failed**, fix the errors shown in the report.

c. Restart the **kea-dhcp4** service:

```
# systemctl restart kea-dhcp4
```

2. If you are configuring an IPv6 network:

a. Edit the **/etc/kea/kea-dhcp6.conf** file, and add the **loggers** configuration to the **Dhcp6** parameter, for example:

```
{
 "Dhcp6": {
  ...,
  "loggers":[
   {
    "name":"kea-dhcp6",
    "output-options":[
      {
       "output":"kea-dhcp6.log",
       "maxsize":104857600,
       "maxver":5
      }
    ],
    "severity":"INFO",
   }
  ],
  ...
```

b. Verify the syntax of the configuration file:

```
# kea-dhcp6 -t /etc/kea/kea-dhcp6.conf
```

If the command returns **Syntax check failed**, fix the errors shown in the report.

c. Restart the **kea-dhcp6** service:

```
# systemctl restart kea-dhcp6
```

Verification

- Monitor the log file you have configured to verify that it displays messages of the expected severity.

3.7. ASSIGNING A STATIC ADDRESS TO A HOST BY USING DHCP

In Kea, you can use a reservation inside a subnet definition to assign a fixed IP address to a media access control (MAC), a DHCP unique identifier (DUID), or other identifiers.

For example, use this method to always assign the same IP address to a server or network device.

Prerequisites

- The **kea-dhcp4** and **kea-dhcp6** services are configured and running.
- You are logged in as the **root** user.

Procedure

1. If you are configuring an IPv4 network:

   a. Edit the **/etc/kea/kea-dhcp4.conf** file, and add a reservation to the **subnet4** parameter:

   ```
   {
     "Dhcp4": {
       "subnet4": [
         {
           "subnet": "192.0.2.0/24",
     ...,
           "reservations": [
             {
               "hw-address": "52:54:00:72:2f:6e",
               "ip-address": "192.0.2.130"
             }
           ],
     ...
   ```

   This example configures Kea to always assign the **192.0.2.130** IP address to the host with the **52:54:00:72:2f:6e** MAC address.

   For further examples, see the **/usr/share/doc/kea/examples/kea4/reservations.json** file provided by the **kea-doc** package.

   b. Verify the syntax of the configuration file:

   ```
   # kea-dhcp4 -t /etc/kea/kea-dhcp4.conf
   ```

   If the command returns **Syntax check failed**, fix the errors shown in the report.

   c. Restart the **kea-dhcp4** service:

```
# systemctl restart kea-dhcp4
```

2. If you are configuring an IPv6 network:

    a. Edit the **/etc/kea/kea-dhcp6.conf** file, and add a reservation to the **subnet6** parameter:

    ```
    {
      "Dhcp6": {
        "subnet6": [
          {
            "subnet": "2001:db8:0:1::/64",
    ...,
            "reservations": [
              {
                "hw-address": "52:54:00:72:2f:6e",
                "ip-address": "2001:db8:0:1::99"
              }
            ];
    ...
    ```

    This example configures Kea to always assign the **2001:db8:0:1::99** IP address to the host with the **52:54:00:72:2f:6e** MAC address.

    For further examples, see the **/usr/share/doc/kea/examples/kea6/reservations.json** file provided by the **kea-doc** package.

    b. Verify the syntax of the configuration file:

    ```
    # kea-dhcp6 -t /etc/kea/kea-dhcp6.conf
    ```

    If the command returns **Syntax check failed**, fix the errors shown in the report.

    c. Restart the **kea-dhcp6** service:

    ```
    # systemctl restart kea-dhcp6
    ```

3.8. CLASSIFYING CLIENTS IN KEA

Kea client classes provide a mechanism for grouping clients based on specific criteria, allowing for granular control over network configuration. You can use this feature to apply special processing rules or assign different DHCP options to clients.

You can create a client class that assigns Voice over IP (VoIP) devices to a specific IP pool to ensures that VoIP phones get different IP addresses than other devices on the network. For example, in IPv4 networks, you can use a **substring** expression to test for the first 3 octets of their media access control (MAC) address. In IPv6 networks where the MAC address is not a reliable indicator, you can test for a substring of the DHCPv6 vendor class option.

Prerequisites

- The **kea-dhcp4** and **kea-dhcp6** services are configured and running.
- You are logged in as the **root** user.

Procedure

1. If you are configuring an IPv4 network:

    a. Edit the **/etc/kea/kea-dhcp4.conf** file, and make the following changes:

        i. Add the following client classes to the **Dhcp4** parameter:

```
{
 "Dhcp4": {
  ...
  "client-classes": [
   {
     "name": "VoIP-Phones",
     "test": "substring(pkt4.mac, 0, 3) == 0x525400"
   },
   {
     "name": "Others",
     "test": "not member('VoIP-Phones')"
   }
  ],
  ...
```

        In this example, devices with a MAC address starting with **52:54:00** match the **VoIP-Phones** client class. Devices that do not match the rule are assigned to the **Others** client class.

        ii. Assign the client classes to your **pool** definitions:

```
{
 "Dhcp4": {
  "subnet4": [
   {
    "subnet": "192.0.2.0/24",
 "pools": [
     {
      "pool": "192.0.2.20  - 192.0.2.100",
      "client-class": "Others"
     },
     {
      "pool": "192.0.2.150 - 192.0.2.200",
      "client-class": "VoIP-Phones"
     }
    ],
    ...
```

        Depending on which client class a host matches, Kea assigns an IP from the corresponding pool.

    b. Verify the syntax of the configuration file:

```
# kea-dhcp4 -t /etc/kea/kea-dhcp4.conf
```

    If the command returns **Syntax check failed**, fix the errors shown in the report.

    c. Restart the **kea-dhcp4** service:

```
# systemctl restart kea-dhcp4
```

2. If you are configuring an IPv6 network:

   a. Edit the **/etc/kea/kea-dhcp6.conf** file, and make the following changes:

      i. Add the following client classes to the **Dhcp6** parameter:

```
{
  "Dhcp6": {
    ...
    "client-classes": [
      {
        "name": "VoIP-Phones",
        "test": "option[16].exists and (substring(option[16].hex, 0, 8) ==
'00000009')",
      },
      {
        "name": "Others",
        "test": "not member('VoIP-Phones')"
      }
    ],
    ...
```

      In this example, devices that send a DHCPv6 vendor class option (option 16) where the hexadecimal value begins with **00000009** match the **VoIP-Phones** client class. Devices that do not match the rule are assigned to the **Others** client class.

      ii. Assign the client classes to your **pool** definitions:

```
{
  "Dhcp6": {
    "subnet6": [
      {
        "subnet": "2001:db8:0:1::/64",
 "pools": [
          {
            "pool": "2001:db8:0:1::1000 - 2001:db8:0:1::2000",
            "client-class": "Others"
          },
          {
            "pool": "2001:db8:0:1::4000 - 2001:db8:0:1::5000",
            "client-class": "VoIP-Phones"
          }
        ],
        ...
```

      Depending on which client class a host matches, Kea assigns an IP from the corresponding pool.

   b. Verify the syntax of the configuration file:

```
# kea-dhcp6 -t /etc/kea/kea-dhcp6.conf
```

   If the command returns **Syntax check failed**, fix the errors shown in the report.

c. Restart the **kea-dhcp6** service:

```
# systemctl restart kea-dhcp6
```

Verification

- Connect clients that match the rules in the client classes and verify that Kea assigned an IP from the associated pool.

3.9. COMPARISON OF DHCPV6 TO RADVD

In an IPv6 network, only router advertisement messages provide information about an IPv6 default gateway. If you want to use DHCPv6 in subnets that require a default gateway setting, you must additionally configure a router advertisement service, such as Router Advertisement Daemon (**radvd**).

The **radvd** service uses flags in router advertisement packets to announce the availability of a DHCPv6 server.

The following table compares features of DHCPv6 and **radvd**:

	DHCPv6	**radvd**
Provides information about the default gateway	no	yes
Guarantees random addresses to protect privacy	yes	no
Sends further network configuration options	yes	no
Maps media access control (MAC) addresses to IPv6 addresses	yes	no

3.10. CONFIGURING THE RADVD SERVICE FOR IPV6 ROUTERS

The router advertisement daemon (**radvd**) sends router advertisement messages that are required for IPv6 stateless autoconfiguration. This enables users to automatically configure their addresses, settings, routes, and to choose a default router based on these advertisements.

> **NOTE**
>
> You can only set /**64** prefixes in the **radvd** service. To use other prefixes, use DHCPv6.

Prerequisites

- You are logged in as the **root** user.

Procedure

1. Install the **radvd** package:

```
# dnf install radvd
```

2. Edit the **/etc/radvd.conf** file, and add the following configuration:

```
interface enp1s0
{
  AdvSendAdvert on;
  AdvManagedFlag on;
  AdvOtherConfigFlag on;

  prefix 2001:db8:0:1::/64 {
  };
};
```

These settings configure **radvd** to send router advertisement messages on the **enp1s0** device for the **2001:db8:0:1::/64** subnet. The **AdvManagedFlag on** setting defines that the client should receive the IP address from a DHCP server, and the **AdvOtherConfigFlag** parameter set to **on** defines that clients should receive non-address information from the DHCP server as well.

For further details, see the **radvd.conf(5)** man page on your system and the **/usr/share/doc/radvd/radvd.conf.example** file.

3. Optional: Configure that **radvd** automatically starts when the system boots:

```
# systemctl enable radvd
```

4. Start the **radvd** service:

```
# systemctl start radvd
```

Verification

- Display the content of router advertisement packages and the configured values **radvd** sends:

```
# radvdump
```

Additional resources

- Can I use a prefix length other than 64 bits in IPv6 Router Advertisements? (Red Hat Knowledgebase)