NOKIA

Nokia Service Router Linux

CONFIGURATION BASICS
RELEASE 22.11

3HE 19030 AAAA TQZZA
Issue 01

November 2022

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Table of contents

1 About this guide....8
1.1 Precautionary and information messages....8
1.2 Conventions....8

2 What's new....10

3 System management....11
3.1 Configuring a hostname....11
3.2 Configuring a domain name....11
3.3 Configuring DNS settings....12
3.4 Configuring the management network-instance....12
3.5 Access types....13
3.5.1 Enabling an SSH server....14
3.5.1.1 Configure SSH key-based authentication....14
3.5.2 Configuring FTP....15
3.6 Configuring banners....16
3.7 Synchronizing the system clock....16
3.8 Configuring SNMP....17
3.9 IP ECMP Load Balancing....18
3.9.1 Configuring IP ECMP load balancing....19

4 Configuration management....20
4.1 Default configuration....20
4.2 Configuration datastores....20
4.3 Configuration modes....20
4.3.1 Configuration candidates....21
4.3.2 Setting the configuration mode....21
4.4 Committing a configuration in candidate mode....22
4.4.1 Confirming a commit operation....23
4.4.2 Validating a commit operation....24
4.4.3 Updating the baseline datastore....24
4.5 Deleting a configuration....25
4.6 Annotating the configuration....25
4.7 Discarding a configuration in candidate mode....26

4.8 Displaying configuration details....27
4.9 Displaying the configuration state....29
4.10 Saving a configuration to a file....30
4.11 Loading a configuration....31
4.12 Executing configuration statements from a file....32
4.13 Configuration checkpoints....32
4.13.1 Generating a checkpoint....33
4.13.2 Loading a checkpoint....33
4.13.3 Reverting to a previous checkpoint....34
4.13.4 Clearing a checkpoint....34
4.13.5 Configuring maximum number of checkpoints....34
4.13.6 Displaying checkpoint information....35
4.14 Rescue configuration....35
4.14.1 Saving a rescue configuration....36
4.14.2 Clearing a rescue configuration....36
4.15 Configuration upgrades....37
4.15.1 Upgrading configuration files....37

5 Securing access....39
5.1 User types....39
5.1.1 Linux users....39
5.1.2 Local users....39
5.1.3 Remote users....39
5.2 AAA functions....40
5.2.1 Authentication....40
5.2.1.1 Password security for local users....40
5.2.2 Authorization....41
5.2.3 Accounting....41
5.3 AAA server group configuration....41
5.3.1 Configuring an AAA server group....42
5.4 Authentication for the linuxadmin user....43
5.4.1 Configuring the password for the linuxadmin user....43
5.5 Authentication for local users....43
5.5.1 Configuring authentication for local users....43
5.5.2 Configuring password security for local users....44
5.5.3 Clear local user lockout....46

5.6 Authorization using role-based access control 47
5.6.1 Role configuration 47
5.6.1.1 Configuring a role 48
5.6.2 Assigning roles to users 50
5.6.3 Authorization using a TACACS+ server 50
5.6.3.1 Configuring TACACS+ Authorization 51
5.6.4 Service authorization for local users 53
5.6.4.1 Configuring service authorization 53
5.7 Accounting configuration 55
5.7.1 Configuring accounting 55
5.8 Displaying user session information 55
5.9 Disconnecting user sessions 56
5.10 Configuring idle-timeout for user sessions 56

6 Management servers 58
6.1 gNMI server 58
6.1.1 Configuring a gNMI server 58
6.2 JSON-RPC server 59
6.2.1 Configuring a JSON-RPC server 60
6.3 TLS profiles 60
6.3.1 Configuring a TLS profile 61
6.3.2 Generating a self-signed certificate 61
6.3.3 Generating a certificate signing request 62

7 Logging 64
7.1 Input sources for log messages 64
7.2 Filters for log messages 66
7.3 Output destinations for log messages 67
7.4 Defining filters 67
7.5 Logging destination configuration 68
7.5.1 Specifying a log file destination 68
7.5.2 Specifying a buffer destination 70
7.5.3 Specifying the console as destination 70
7.5.4 Specifying a remote server destination 71
7.6 Specifying a Linux syslog facility for SR Linux subsystem messages 71

8 Network-instances 73
8.1 Basic network-instance configuration 73
8.2 Path MTU discovery 74
8.3 Static routes 74
8.3.1 Configuring static routes 75
8.3.2 Configuring failure detection for static routes 76
8.4 Aggregate routes 76
8.4.1 Configuring aggregate routes 77
8.5 Route preferences 78
8.6 Displaying network-instance status 78
8.7 mac-vrf network-instance 79
8.7.1 MAC selection 80
8.7.2 MAC duplication detection and actions 80
8.7.2.1 MAC duplication detection 80
8.7.2.2 MAC duplication actions 80
8.7.2.3 MAC duplication process restarts 81
8.7.2.4 Configurable hold-down-time 81
8.7.3 Bridge table configuration 81
8.7.3.1 Deleting entries from the bridge table 81
8.7.4 mac-vrf network instance configuration 82

9 BFD 84
9.1 Configuring BFD for a subinterface 84
9.2 Configuring BFD under the BGP protocol 85
9.3 Configuring BFD under OSPF 86
9.4 Configuring BFD under IS-IS 86
9.5 Viewing the BFD state 87
9.6 Micro-BFD 87
9.6.1 Configuring micro-BFD for a LAG interface 88
9.6.2 Viewing the micro-BFD state 88

10 SR Linux applications 90
10.1 Installing an application 90
10.2 Starting an application 91
10.3 Restarting applications 91

10.3.1 Restarting an application ... 92
10.4 Terminating an application ... 93
10.5 Reloading application configuration ... 93
10.6 Clearing application statistics ... 94
10.7 Restricted operations for applications ... 94
10.8 Configuring an application ... 95
10.9 Removing an application from the system ... 99
10.10 Partioning and isolating application resources ... 100
10.10.1 Cgroup profiles ... 100
10.10.1.1 Default cgroup profile ... 100
10.10.1.2 Customer-defined cgroup profile ... 103
10.10.2 Configuring a cgroup ... 103
10.10.2.1 Cgroup configuration example ... 103
10.10.3 Kernel low-memory killer ... 105
10.10.3.1 SR Linux process kill strategy ... 105
10.10.4 Application manager extensions for cgroups ... 108
10.10.5 Debugging cgroups ... 108
10.10.5.1 SR Linux cgroup debugging commands ... 109
10.10.5.2 Linux-provided cgroup debugging commands ... 112

11 UFT profiles ... 115
11.1 Shared bank partitioning for SR Linux systems ... 115
11.2 LPM table partitioning ... 116
11.3 Default UFT allocations for SR Linux systems ... 116
11.4 Configuring a UFT profile ... 117
11.5 Displaying UFT profile information ... 117

12 Maintenance Mode ... 119
12.1 Configuring a maintenance group ... 119
12.2 Configuring a maintenance profile ... 120
12.3 Placing a maintenance group into maintenance mode ... 121
12.4 Taking a maintenance group out of service ... 121
12.5 Restoring the maintenance group to service ... 122

1 About this guide

This document describes basic configuration for the Nokia Service Router Linux (SR Linux). Examples of commonly used commands are provided.

This document is intended for network technicians, administrators, operators, service providers, and others who need to understand how the router is configured.

Note:
This manual covers the current release and may also contain some content that will be released in later maintenance loads. See the *SR Linux Release Notes* for information on features supported in each load.

1.1 Precautionary and information messages

The following are information symbols used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.2 Conventions

Nokia SR Linux documentation uses the following command conventions.

- **Bold** type indicates a command that the user must enter.
- Input and output examples are displayed in `Courier` text.
- An open right-angle bracket indicates a progression of menu choices or simple command sequence (often selected from a user interface). Example: **start** > **connect to**.
- A vertical bar (|) indicates a mutually exclusive argument.
- Square brackets ([ ]) indicate optional elements.

- Braces ({ }) indicate a required choice. When braces are contained within square brackets, they indicate a required choice within an optional element.
- *Italic* type indicates a variable.

Generic IP addresses are used in examples. Replace these with the appropriate IP addresses used in the system.

2 What's new

Topic	Location
Password security for local users	Password security for local users Configuring password security for local users
Configurable password for the SR Linux `linuxadmin` user	Authentication for the linuxadmin user
Application warm restart	Restarting applications
ACLs and policy-based routing	ACLs and policy-based routing has been moved to the new *SR Linux ACL and Policy-Based Routing Guide*
Interfaces	Interface functionality has been moved to the new *SR Linux Interfaces Guide*
Routing protocols	Routing protocol functionality (BGP, IS-IS, OSPF) has been moved to the new *SR Linux Routing Protocols Guide*
Mirroring	Mirroring functionality has been moved to the *SR Linux Troubleshooting Toolkit Guide*

3 System management

This chapter contains procedures for setting up basic system management functions on SR Linux, including the hostname, domain name, DNS settings, and management network-instance. It contains examples of configuring an SSH server and FTP server, as well as NTP for the system clock, and enabling an SNMP server.

3.1 Configuring a hostname

Procedure

The SR Linux device must have a hostname configured. The default hostname is `srlinux`. The hostname normally appears on all CLI prompts on the device, although you can override this with the **environment prompt** CLI command.

The hostname should be a unique name on the network, and can be a fully qualified domain name (FQDN), or an unqualified single-label name. If the hostname is a single-label name (for example, `srlinux`), the system may use its domain name, if configured, to infer its own FQDN.

Example:

The following example shows the configuration for a hostname on the SR Linux device.

```
--{ candidate shared default }--[  ]--
# info system name
    system {
        name {
            host-name 3-node_srlinux-A
        }
    }
```

3.2 Configuring a domain name

Procedure

The SR Linux device uses its hostname, combined with a domain name to form its fully qualified domain name (FQDN). It is expected that the FQDN exists within the DNS servers used by SR Linux, though this is not a requirement.

Assuming the SR Linux FQDN is in the DNS server, you can use the FQDN to reach the SR Linux device without knowing its management address. A domain name is not mandatory, but if specified, it is added to the DNS search list by default.

Example:

The following shows the configuration for a domain name on the SR Linux device. In this example, the device FQDN is set to `3-node_srlinux-A.mv.usa.nokia.com`.

```
--{ candidate shared default }--[  ]--
```

```
# info system name
    system {
        name {
            host-name 3-node_srlinux-A
            domain-name mv.usa.nokia.com
        }
    }
```

3.3 Configuring DNS settings

Procedure

The SR Linux device uses DNS to resolve hostnames within the configuration, or for operational commands, such as ping. You can specify up to three DNS servers for the SR Linux device to use, with either IPv4 or IPv6 addressing.

You can also specify a search list of DNS suffixes that the device can use to resolve single-label names; for example, for a search list of `nokia1.com` and `nokia2.com`, a ping for host `srlinux` does a DNS lookup for `srlinux.nokia1.com`, and if unsuccessful, does a DNS lookup for `srlinux.nokia2.com`.

The SR Linux device supports configuration of static DNS entries. Static DNS entries allow resolution of hostnames that may not be in the DNS servers used by the SR Linux device. Using a static DNS entry, you can map multiple addresses (both IPv4 and IPv6) to one hostname. The SR Linux linux_mgr application adds the static DNS entries to the `/etc/hosts` file in the underlying Linux OS.

Example:

In the following example, the SR Linux device is configured to use two DNS servers to resolve hostnames, a search list of DNS suffixes for resolving single-label names, and IPv4 and IPv6 static DNS entries for a host.

DNS requests are sourced from the mgmt network-instance (see Configuring the management network-instance).

```
--{ candidate shared default }--[  ]--
# info system dns
    system {
        dns {
            network-instance mgmt
            server-list [
                1.1.1.1
                2.2.2.2
            ]
            search-list [
                nokia1.com
                nokia2.com
            ]
            host-entry srlinux.nokia.com {
                ipv4-address 3.3.3.3
                ipv6-address 2001:db8:123:456::11:11
            }
        }
    }
```

3.4 Configuring the management network-instance

Procedure

Management of the SR Linux device is primarily done via a management network-instance. The management network-instance isolates management traffic from other network-instances configured on the device.

The out-of-band `mgmt0` interface is automatically added to the management network-instance, and management services run within the management network-instance.

Although the management network-instance is primarily intended to handle management traffic, you can configure it in the same way as any other network-instance on the device, including protocols, policies, and filters. The management network instance is part of the default configuration, but may be deleted if necessary.

Addressing within the management network-instance is available via DHCP and static IP addresses. Both IPv4 and IPv6 addresses are supported.

Example:

```
--{ candidate shared default }--[  ]--
# info network-instance mgmt
    network-instance mgmt {
        type ip-vrf
        admin-state enable
        description "Management network instance"
        interface mgmt0.0 {
        }
        protocols {
            linux {
                export-routes true
                export-neighbors true
            }
        }
    }
```

3.5 Access types

Access to the SR Linux device is available via a number of APIs and protocols. The SR Linux supports the following ways to access the device:

- SSH – Secure Shell, a standard method for accessing network devices. See Enabling an SSH server.
- FTP – File Transfer Protocol, a secure method for transferring files to and from network devices. See Configuring FTP.
- Console – Access to the SR Linux CLI via direct connection to a serial port on the device.
- gNMI – A gRPC-based protocol for the modification and retrieval of configuration from a target device, as well as the control and generation of telemetry streams from a target device to a data collection system. See gNMI server.
- JSON-RPC – Ability to retrieve and set configuration and state using a JSON-RPC API. See JSON-RPC server.
- SNMP – Simple Network Management Protocol, a commonly used network management protocol. The SR Linux device supports SNMPv2 with a limited set of OIDs.

Regardless of the method of access, all sessions are authenticated (if authentication is enabled), whether the session is entered via the console, SSH, or an API. Access to the device is controlled via the aaa_mgr application. See Securing access.

3.5.1 Enabling an SSH server

Procedure

You can enable an SSH server for one or more network instances on the SR Linux device, so that users can log in to the CLI using an SSH client. The SR Linux device implements SSH via OpenSSH, and configures `/etc/ssh/sshd_config` in the underlying Linux OS. Only SSHv2 is supported.

Example:

In the following example, an SSH server is enabled in the mgmt and default network-instances, specifying the IP addresses where the device listens for SSH connections:

```
--{ candidate shared default }--[  ]--
# info system ssh-server
    system {
        ssh-server {
            network-instance mgmt {
                admin-state enable
                source-address [
                    1.1.1.1
                    1.1.1.2
                ]
            }
            network-instance default {
                admin-state enable
                source-address [
                    2.1.1.1
                    2.1.1.2
                ]
            }
        }
    }
```

3.5.1.1 Configure SSH key-based authentication

Procedure

The SR Linux SSH server supports RSA public-private key-based authentication, where an SSH client provides a signed message that has been encrypted by a private key. If the SSH client's corresponding public key is configured on the SR Linux, the SSH server can authenticate the client.

When performing authentication for a user, the SR Linux first tries public-key authentication; if this fails, the SR Linux tries password authentication.

To configure SSH key-based authentication, you generate a public-private key pair, then add the public key to the SR Linux.

Example:

The following is an example of using the ssh-keygen utility in Linux to generate an RSA key pair with a length of 2,048 bits:

```
# ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:RNVV8/XRVK7PhY2OJxa7rjkSUFqyVoj4pUXL2PDs7mI user@linux
The key's randomart image is:
+---[RSA 2048]----+
| .o+o. ...oo*|
| o.oB*.. +=|
| . .o@* +|
| Fo = |
| . .M . + o|
| .. = o.|
| .. = o o|
| E.. .o + |
| . ...o+o |
+----[SHA256]-----+
```

Example

After generating the RSA key pair, you can add the public key to the SR Linux. The location for the public key depends on the type of user for which SSH key-based authentication is being configured:

- For Linux users (see Linux users), you add the public key to the user's `$HOME/.ssh/authorized_keys` file.
- For users configured within the SR Linux CLI (see Local users), you add the public key to the SR Linux configuration file. This can be done with a CLI command.

For example, the following CLI command configures a public key and password for the SR Linux user `srlinux`:

```
--{ candidate shared default }--[  ]--
# system aaa authentication user username srlinux ssh-key
[ <public-key> ] password <password>
```

In the example, the `<public-key>` has the format `ssh-rsa <key> <comment>`. If multiple public keys are configured for a user, they are tried in the order they were configured.

3.5.2 Configuring FTP

Procedure

You can enable an FTP server for one or more network instances on the SR Linux device, so that users can transfer files to and from the device. The SR Linux uses the vsftpd (very secure FTP daemon) application within the underlying Linux OS. The authenticated user's home directory returned by the aaa_mgr application is set as the user's FTP root directory.

Example:

In the following example, the FTP server is enabled in the mgmt and default network-instance, specifying the IP addresses where the device listens for FTP connections:

```
--{ candidate shared default }--[  ]--
# info system ftp-server
    system {
        ftp-server {
            network-instance mgmt {
                admin-state enable
                source-address [
                    1.1.1.1
                ]
            }
            network-instance default {
                admin-state enable
                source-address [
                    2.1.2.1
                ]
            }
        }
    }
```

3.6 Configuring banners

Procedure

You can specify banner text that appears when a user connects to the SR Linux device. The following banners can be configured:

- Login banner – Displayed before a user has been authenticated by the system (for example, at the SSH login prompt)
- Message of the day (motd) banner – Displayed after the user has been authenticated by the system

The banners appear regardless of the method used to connect to the SR Linux, so they are displayed to users connecting via SSH, console, and so on.

Example:

In the following example, login and motd banners are configured. The login banner text appears at the prompt when a user attempts to log in to the system, and the motd banner text appears after the user has been authenticated.

```
--{ candidate shared default }--[  ]--
# info system banner
    system {
        banner {
            login-banner "Enter your SRLinux login credentials."
            motd-banner "Welcome to the SRLinux CLI. Your activity may be monitored."
        }
    }
```

3.7 Synchronizing the system clock

Procedure

Network Time Protocol (NTP) is used to synchronize the system clock to a time reference. You can configure NTP settings on the SR Linux device using the CLI, and the SR Linux linux_mgr application provisions the settings in the underlying Linux OS.

NTP does not account for time zones, instead relying on the host to perform such computations. Time zones on the SR Linux device are based on the IANA tz database, which is implemented by the underlying Linux OS. You can specify the time zone of the SR Linux device using the CLI.

Example:

The following configuration enables the system NTP client on the SR Linux device and specifies an NTP server to use for clock synchronization. The NTP client runs in the mgmt network-instance. The system time zone is set to America/Los_Angeles.

```
--{ candidate shared default }--[  ]--
# info system ntp
    system {
        ntp {
            admin-state enable
            network-instance mgmt
            server 4.53.160.75 {
            }
        }
        clock {
            timezone America/Los_Angeles
        }
    }
```

3.8 Configuring SNMP

Procedure

The SR Linux device supports SNMPv2. See the *SR Linux System Management Guide* for descriptions of the supported SNMP OIDs. The MIB file that covers these OIDs is packaged with each release.

Example:

In the following example, an SNMP server is running within the mgmt and default network-instances, and the configuration specifies the IP addresses where the device listens for SNMP client connections:

```
--{ candidate shared default }--[  ]--
# info system snmp
    system {
        snmp {
            community test1 {
                permission r
                version v2c
            }
            network-instance mgmt {
                admin-state enable
                source-address [
                    1.1.1.1
```

```
                ]
            }
            network-instance default {
                admin-state enable
                source-address [
                    3.3.3.3
                ]
            }
        }
    }
```

3.9 IP ECMP Load Balancing

Equal-Cost Multipath Protocol (ECMP) refers to the distribution of packets over two or more outgoing links that share the same routing cost. Static, IS-IS, OSPF, and BGP routes to IPv4 and IPv6 destinations can be programmed into the datapath by their respective applications, with multiple IP ECMP next-hops.

The SR Linux load-balances traffic over multiple equal-cost links with a hashing algorithm that uses header fields from incoming packets to calculate which link to use. When an IPv4 or IPv6 packet is received on a subinterface, and it matches a route with a number of IP ECMP next-hops, the next-hop that forwards the packet is selected based on a computation using this hashing algorithm. The goal of the hash computation is to keep packets in the same flow on the same network path, while distributing traffic proportionally across the ECMP next-hops, so that each of the *N* ECMP next-hops carries approximately 1/*N*th of the load.

The hash computation takes various key and packet header field values as inputs and returns a value that indicates the next-hop. The key and field values that can be used by the hash computation depend on the platform, packet type, and configuration options, as follows:

On 7250 IXR systems, the following can be used in the hash computation:

- For IPv4 TCP/UDP non-fragmented packets: user-configured hash-seed (0-65535; default 0), source IPv4 address, destination IPv4 address, IP protocol, L4 source port, L4 destination port. The algorithm is asymmetric; that is, inverting source and destination pairs does not produce the same result.
- For IPv6 TCP/UDP non-fragmented packets: user-configured hash-seed (0-65535; default 0), source IPv6 address, destination IPv6 address, IPv6 flow label (even if it is 0), IP protocol (IPv6 next-header value in the last extension header), L4 source port, L4 destination port. The algorithm is symmetric; that is, inverting source and destination pairs produces the same result.
- For all other packets: user-configured hash-seed (0-65535; default 0), source IPv4 or IPv6 address, destination IPv4 or IPv6 address.

On 7220 IXR-D1, D2, D3 and 7220 IXR-H2 and H3 systems, the following can be used in the hash computation:

- For IPv4 TCP/UDP non-fragmented packets: VLAN ID, user-configured hash-seed (0-65535; default 0), source IPv4 address, destination IPv4 address, IP protocol, L4 source port, L4 destination port. The algorithm is asymmetric.
- For IPv6 TCP/UDP non-fragmented packets: VLAN ID, user-configured hash-seed (0-65535; default 0), source IPv6 address, destination IPv6 address, IPv6 flow label (even if it is 0), IP protocol (IPv6 next-header value in the last extension header), L4 source port, L4 destination port.
- For all other packets: user-configured hash-seed (0-65535; default 0), source IPv4 or IPv6 address, destination IPv4 or IPv6 address.

3.9.1 Configuring IP ECMP load balancing

Procedure

To configure IP ECMP load balancing, you specify hash-options that are used as input fields for the hash calculation, which determines the next-hop for packets matching routes with multiple ECMP hops.

Example:

The following example configures hash options for IP ECMP load balancing, including a hash seed and packet header field values to be used in the hash computation.

```
--{ * candidate shared default }--[  ]--
# info system load-balancing
    system {
        load-balancing {
            hash-options {
                hash-seed 128
                ipv6-flow-label false
            }
        }
    }
```

If no value is configured for the **hash-seed** the default value is 0. If a hash-option is not specifically configured, the default is **true**.

On 7250 IXR systems, if **source-address** is configured as a hash option, the **destination-address** must also be configured as a hash option. Similarly, if **source-port** is configured as a hash option, the **destination-port** must also be configured as a hash option.

4 Configuration management

The chapter describes concepts behind managing the SR Linux configuration, including configuration datastores, modes, and how to commit changes to the running configuration.

4.1 Default configuration

At startup, the SR Linux loads a JSON configuration file, located at `/etc/opt/srlinux/config.json`. If this startup configuration does not exist, the system is started using a factory default configuration.

The factory default configuration brings device into management, enables DHCP/v6 on the management interface, adds it to the management network-instance, enables an SSH server, and creates various system logs and applies a default set of CPM filters.

You can optionally create a rescue configuration, which is loaded if the startup configuration fails to load (see Rescue configuration). If the startup configuration fails to load, and no rescue configuration exists, the system is started using the factory default configuration.

4.2 Configuration datastores

Configuration and state information reside in datastores on the SR Linux device. The following datastores are available:

- Running – contains the currently active configuration.
- State – contains the running configuration, plus dynamically added data such as operational state of interfaces or BGP peers added via auto-discovery, as well as session states and routing tables.
- Candidate – contains a user-configurable version of the running datastore. After it has been committed, the candidate datastore becomes the running datastore.
- Tools – contains executable commands that allow you to perform operations such as restarting the device and clearing interface statistics.

Within the CLI, you can use the **info** command to display information from a datastore. For example, entering the **info from state** command (or entering the **info** command in state mode) displays configuration and statistics from the state datastore for the current context, and entering the **info from running** command (or the **info** command in running mode) displays configuration from the running datastore for the current context.

4.3 Configuration modes

The candidate datastore corresponds to a configuration mode within the SR Linux CLI. In candidate mode, you can modify SR Linux configuration settings.

By default, candidate mode operates in shared mode, which allows multiple users to modify the candidate configuration concurrently. When the configuration is committed in shared mode, all of the users' changes are applied.

You can optionally use candidate mode in exclusive mode, which locks out all other users from making changes to the candidate configuration.

4.3.1 Configuration candidates

When a user enters candidate mode, the system creates two copies of the running datastore: one is modifiable by the user, and the other serves as a baseline. The modifiable datastore and the baseline datastore are collectively known as a configuration candidate.

A configuration candidate can be either shared or private.

- Shared – is the default configuration candidate for CLI sessions. Multiple users can modify the shared candidate concurrently. When the configuration is committed, the changes from all of the users are applied.
- Private – is the default configuration candidate when using JSON-RPC or gNMI clients, and can optionally be used in the CLI. With a private candidate, each user modifies their own separate instance of the configuration candidate. When a user commits their changes, only the changes from that user are committed.

By default, there is a single unnamed global configuration candidate. You can optionally configure one or more named configuration candidates, which function identically to the global configuration candidate. Both shared and private configuration candidates support named versions.

4.3.2 Setting the configuration mode

Procedure

After logging in to the CLI, you are initially placed in running mode. Table 1: Commands to change configuration mode describes the commands to change between modes.

Table 1: Commands to change configuration mode

To enter this mode:	Type this command:
Candidate shared	**enter candidate**
Candidate mode for named shared candidate	**enter candidate name <name>**
Candidate private	**enter candidate private**
Candidate mode for named private candidate	**enter candidate private name <name>**
Candidate exclusive	**enter candidate exclusive**
Exclusive mode for named candidate	**enter candidate exclusive name <name>**
Running	**enter running**

To enter this mode:	Type this command:
State	**enter state**
Show	**enter show**

Example:

For example, to change from running to candidate mode:

```
--{ running }--[  ]--
# enter candidate
--{ * candidate shared default }--[  ]--
```

The asterisk (*) next to the mode name indicates that the candidate configuration has changes that have not yet been committed.

Example:

To enter candidate mode for a named configuration candidate, you specify the name of the configuration candidate. For example:

```
--{ running }--[  ]--
# enter candidate name cand1
--{ candidate shared cand1}--[  ]--
```

4.4 Committing a configuration in candidate mode

About this task

Changes made during a configuration modification session do not take effect until a **commit** command is issued. Use the **commit** command in candidate mode only.

Procedure

Step 1. Enter candidate mode:

Example

```
# enter candidate
```

Step 2. Enter configuration commands.

Step 3. Enter the **commit** command:

- To apply the changes and remain in candidate mode, enter **commit stay**.
- To apply the changes, exit candidate mode, and enter running mode, enter **commit now**.
- To apply the changes, remain in candidate mode, and save the changes to the startup configuration, enter **commit stay save**.
- To apply a comment to a **commit stay** or **save** operation, use the **comment** keyword and specify a comment.

Example

```
# enter candidate
--{ candidate shared default }--[  ]--
# interface ethernet-1/1 subinterface 1
--{ * candidate shared default }--[ interface ethernet-1/1 subinterface 1 ]--
# commit stay
All changes have been committed. Starting new transaction.
--{ + candidate shared default }--[ interface ethernet-1/1 subinterface 1 ]--
# save startup
/system:
    Saved current running configuration as initial (startup) configuration '/etc/
opt/srlinux/config.json'
--{ candidate shared default }--[ interface ethernet-1/1 subinterface 1 ]--
#
```

In this example, a user enters candidate mode and creates a subinterface for interface `ethernet-1/1`. The asterisk (*) next to `candidate shared default` in the prompt indicates that the candidate configuration has changes that have not yet been committed. After committing the changes with the **commit stay** command, the new subinterface becomes part of the running configuration.

The plus sign (+) in the prompt indicates that the currently running configuration differs from the startup configuration. The **save startup** command saves the running configuration to the startup configuration.

4.4.1 Confirming a commit operation

Procedure

You can optionally configure the SR Linux to require explicit confirmation via a **tools** command for the configuration changes from a commit operation to become permanent. If the new configuration is not confirmed after a timeout period, the running datastore reverts to the previous version.

Example:

After entering configuration commands in candidate mode, use the following command to commit the configuration and start the confirmation timer:

```
--{ candidate shared default }--[  ]--
# commit confirmed
```

The **commit confirmed** command applies the changes to the running datastore and activates them. If the configuration is committed successfully, the confirmation timer is started (default 10 minutes). If you do not confirm the commit operation before the timer expires, the configuration reverts to the previous version.

Example

To confirm the commit operation and make the configuration changes permanent, enter the following command before the confirmation timer expires:

```
--{ candidate shared default }--[  ]--
# tools system configuration confirmed-accept
```

Example

To revert to the previous configuration without waiting for the confirmation timer to expire, enter the following command:

```
--{ candidate shared default }--[  ]--
# tools system configuration confirmed-reject
```

4.4.2 Validating a commit operation

Procedure

You can optionally validate the configuration changes made during a commit operation before they are applied to the running datastore. The SR Linux management server checks the syntax of the changes in the candidate configuration and displays messages if validation errors are found.

Example:

Use the following command to perform a validation check for configuration changes. Note that this command only validates the changes; it does not apply them to the running datastore.

```
--{ candidate shared default }--[  ]--
# commit validate
```

If syntax errors are found in the configuration changes, SR Linux displays error messages; if validation is successful, no output is displayed.

4.4.3 Updating the baseline datastore

Procedure

A configuration candidate consists of a modifiable candidate datastore and a baseline datastore, both of which are snapshots of the then-current running datastore when a user enters candidate mode.

During the lifetime of the configuration candidate, the running datastore can be modified via commits initiated from other configuration sessions. At that point, the baseline in the configuration candidate is out-of-date.

Updating the baseline datastore takes a new snapshot of the running configuration, applies the changes in the candidate datastore, and checks for configuration conflicts in the updated baseline datastore. If conflicts are found, the user is informed with a warning or error for each conflict.

A baseline datastore update is done automatically when the user commits the changes to the configuration, or can be done manually with the **baseline update** command.

Example:

In the following example, the baseline datastore in a configuration candidate is out of date, as indicated by the ! in the prompt. This indicates that another user has committed changes to the running datastore.

Entering the **baseline update** command copies the current running datastore to the baseline datastore, applies the changes in the candidate datastore, then displays any conflicts in the updated baseline datastore. If there are no conflicts, no output is returned by the command.

```
--{!* candidate shared default }--[  ]--
# baseline update
--{* candidate shared default }--[  ]--
#
```

4.5 Deleting a configuration

Procedure

Use the **delete** command to delete configurations while in candidate mode.

Example:

The following example displays the system banner configuration, deletes the configured banner, then displays the resulting system banner configuration:

```
--{ candidate shared default }--[  ]--
# info system banner
    system {
        banner {
            login-banner "Welcome to SRLinux!"
        }
    }
--{ candidate shared default }--[  ]--
# delete system banner
--{ candidate shared default }--[  ]--
# info system banner
    system {
        banner {
        }
    }
```

4.6 Annotating the configuration

Procedure

To aid in reading a configuration, you can add comments or descriptive annotations. The annotations are indicated by ! ! ! in displayed output.

You can enter a comment either directly from the command line or by navigating to a CLI context and entering the comment in annotate mode.

Example:

The following example adds a comment to an ACL configuration. If there is already a comment in the configuration, the new comment is appended to the existing comment.

```
--{ candidate shared default }--[  ]--
# acl ipv4-filter ip_tcp !! "Filter TCP traffic"
```

Example

To replace the existing comment, use **!!!** instead of **!!** in the command.

The following example adds the same comment to the ACL by navigating to the context for the ACL and entering the comment in annotate mode:

```
--{ * candidate shared default }--[  ]--
# acl ipv4-filter ip_tcp
--{ * candidate shared default }--[ acl ipv4-filter ip_tcp ]--
# annotate
Press [Meta+enter] or [Esc] followed by [Enter] to finish
-> Filter TCP traffic
```

You can enter multiple lines in annotate mode. To exit annotate mode, press Esc, then the Enter key.

Example

In CLI output, the comment is displayed in the context it was entered. For example:

```
--{ running }--[  ]--
# info acl
    acl {
        ipv4-filter ip_tcp {
            !!! Filter TCP traffic
            entry 100 {
                action {
                    drop {
                        log true
                    }
                }
            }
            entry 110 {
                action {
                    accept {
                        log true
                    }
                }
            }
        }
    }
```

To remove a comment, enter annotate mode for the context and press Esc then Enter without entering any text.

4.7 Discarding a configuration in candidate mode

Procedure

You can discard previously applied configurations with the **discard** command. Use the **discard** command in candidate mode only.

- To discard the changes and remain in candidate mode with a new candidate session, enter **discard stay**.
- To discard the changes, exit candidate mode, and enter running mode, enter **discard now**.

Example:

```
All changes have been committed. Starting new transaction.

--{ candidate shared default }--[ interface ethernet-1/1 subinterface 1 ]--
# discard stay
--{ candidate shared default }--[ interface ethernet-1/1 subinterface 1 ]--
```

4.8 Displaying configuration details

Procedure

The **info** command displays configuration and state information. Entering the **info** command from the root context displays the entire configuration, or the configuration for a specified context. Entering the command from within a context limits the display to the configuration under that context. Use this command in candidate or running mode.

Example:

To display the entire configuration, enter **info** from the root context:

```
--{ candidate shared default }--[  ]--
# info
<all the configuration is displayed>
--{ candidate }--[  ]--
```

Example

To display the configuration for a specific context, enter **info** and specify the context:

```
--{ candidate shared default }--[  ]--
# info system lldp
    system {
        lldp {
            admin-state enable
            hello-timer 600
            management-address mgmt0.0 {
                type [
                    IPv4
                ]
            }
            interface mgmt0 {
                admin-state disable
            }
        }
    }
--{ candidate }--[  ]--
```

Example

From a context, use the **info** command to display the configuration under that context:

```
--{ candidate shared default }--[  ]--
# system lldp
--{ candidate }--[ system lldp ]--
# info
    admin-state enable
```

```
    hello-timer 600
    management-address mgmt0.0 {
        type [
            IPv4
        ]
    }
    interface mgmt0 {
        admin-state disable
    }
--{ candidate }--[ system lldp ]--
```

Example

Use the **as json** option to display JSON-formatted output:

```
--{ candidate }--[ system lldp ]--
# info | as json
{
  "admin-state": "enable",
  "hello-timer": "600",
  "management-address": [
    {
      "subinterface": "mgmt0.0",
      "type": [
        "IPv4"
      ]
    }
  ],
  "interface": [
    {
      "name": "mgmt0",
      "admin-state": "disable"
    }
  ]
}
```

Example

Use the **detail** option to display values for all parameters, including those not specifically configured:

```
--{ candidate }--[ system lldp ]--
# info detail
    admin-state enable
    hello-timer 600
    hold-multiplier 4
    management-address mgmt0.0 {
        type [
            IPv4
        ]
    }
    interface mgmt0 {
        admin-state disable
    }
```

Example

Use the **flat** option to display the output as a series of **set** statements, omitting indentation for any sub-contexts:

```
--{ candidate }--[ system lldp ]--
# info flat
set / system lldp admin-state enable
```

```
set / system lldp hello-timer 600
set / system lldp management-address mgmt0.0
set / system lldp management-address mgmt0.0 type [ IPv4 ]
set / system lldp interface mgmt0
set / system lldp interface mgmt0 admin-state disable
```

Example

Use the **depth** option to display parameters with a specified number of sub-context levels:

```
--{ candidate }--[ system lldp ]--
# info depth 0
    admin-state enable
    hello-timer 600
```

```
--{ candidate }--[ system lldp ]--
# info depth 1
    admin-state enable
    hello-timer 600
    management-address mgmt0.0 {
        type [
            IPv4
        ]
    }
    interface mgmt0 {
        admin-state disable
    }
```

4.9 Displaying the configuration state

Procedure

To display information from the state datastore, enter the **info from state** command in candidate or running mode, or the **info** command in state mode.

Example:

To display state information for a specified context from candidate or running mode:

```
--{ candidate shared default }--[  ]--
# info from state routing-policy policy bgp-export-policy
    routing-policy {
        policy bgp-export-policy {
            statement 999 {
                action {
                    accept {
                    }
                }
            }
        }
    }
```

Example

To display state information for a specified context from state mode:

```
--{ candidate shared default }--[  ]--
```

```
# enter state
--{ state }--[  ]--
# info routing-policy policy bgp-export-policy
    routing-policy {
        policy bgp-export-policy {
            statement 999 {
                action {
                    accept {
                    }
                }
            }
        }
    }
--{ state }--[  ]--
```

Example

You can change to a different mode (for example, from state mode to candidate mode), and remain in the previous context. For example:

```
--{ candidate shared default }--[  ]--
# enter state
--{ state }--[  ]--
# routing-policy policy bgp-export-policy
--{ state }--[ routing-policy policy bgp-export-policy ]--
# info
    statement 999 {
        action {
            accept {
            }
        }
    }
--{ state }--[ routing-policy policy bgp-export-policy ]--
# enter candidate
--{ candidate shared default }--[ routing-policy policy bgp-export-policy ]--
```

4.10 Saving a configuration to a file

Procedure

Save the existing configuration to a file using the **save** command. Use this command in candidate or running mode.

Example:

To save the running configuration to a file:

```
--{ running }--[  ]--
# save file running-config.txt text from running
/ running configuration has been stored in 'running-config.txt'
--{ running }--[  ]--
```

Example

To save the running configuration in JSON format:

```
--{ running }--[  ]--
# save file running-config.json from running
```

```
/ running configuration has been stored in 'running-config.json'
--{ running }--[  ]--
```

Example

To save the running configuration in JSON format:

```
--{ running }--[  ]--
# save file running-config.json from running
/ running configuration has been stored in 'running-config.json'
--{ running }--[  ]--
```

Example

To save the running configuration to the initial (startup) configuration:

```
--{ + running }--[  ]--
# save startup
/ running configuration has been stored in '/etc/opt/srlinux/config.json'
--{ running }--[  ]--
```

The plus sign (+) in the prompt indicates that the running configuration differs from the startup configuration. After you enter the **save startup** command, the running configuration is synchronized with the startup configuration, and the plus sign is removed from the prompt.

4.11 Loading a configuration

Procedure

Use the **load** command to load a configuration. The configuration can be from a checkpoint (see Configuration checkpoints), a JSON-formatted configuration file, the startup configuration, the factory default configuration, a rescue configuration (see Rescue configuration), or from manually entered or pasted JSON-formatted input.

Example:

To load a configuration from a checkpoint:

```
--{ * candidate shared default }--[  ]--
# load checkpoint id 0
/system/configuration/checkpoint[id=0]:
    Loaded checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-0.json'
```

Example

To load a configuration from a file:

```
--{ * candidate shared default }--[  ]--
# load file home/config.txt
Loading configuration from 'home/config.txt'
```

Example

To load a rescue JSON configuration from a checkpoint:

```
--{ * candidate shared default }--[  ]--
```

```
# load rescue auto-commit
/system/configuration/checkpoint[id=__rescue__]:
    Reverting to rescue configuration
```

Example

To load a configuration from manually entered JSON-formatted input:

```
--{ * candidate shared default }--[  ]--
# system banner
--{ * candidate shared default }--[ system banner ]--
# load json
Press [Meta+enter] or [Esc] followed by [Enter] to finish
<< {
<<   "login-banner": "Welcome to SRLinux!"
<< }
```

You can enter or paste multiple lines at the << prompt in JSON-input mode. To exit JSON-input mode, press Esc, then the Enter key.

4.12 Executing configuration statements from a file

Procedure

You can execute configuration statements from a source file consisting of **set** statements such as those generated by the **info flat** command (see Displaying configuration details). The SR Linux reads the file and executes each configuration statement line-by-line. You can optionally commit the configuration automatically after the file is read.

Example:

The following example executes a configuration from a specified file:

```
--{ running }--[  ]--
# source config.cfg
Sourcing commands from 'config.cfg'
Executed 20 lines in 1.6541 seconds from file config.cfg
```

Use the **auto-commit** option to commit the configuration after the commands in the source file are executed.

4.13 Configuration checkpoints

You can roll back the configuration to a previous state, known as a checkpoint. You can load a saved checkpoint into the candidate configuration, and revert the running configuration to a previously saved checkpoint.

A checkpoint is saved as a JSON-formatted file containing the complete configuration for the system. If a checkpoint file is larger than 1 Mb, it is compressed and saved in GZIP format. Checkpoint files are saved in the `/etc/opt/srlinux/checkpoint` directory. They are named `checkpoint-<number>.json` (for example, `checkpoint-0.json`), or `checkpoint-<number>.json.gz`, with the lowest number being the most recently saved checkpoint.

By default, the 10 most recent checkpoints are saved; you can configure the maximum number of checkpoint files that are kept by the system.

4.13.1 Generating a checkpoint

Procedure

You can generate a checkpoint with a **tools** command or with the **save checkpoint** command. You can optionally configure the system to generate a checkpoint automatically when a configuration is committed.

Example:

The following example generates a checkpoint from the current configuration:

```
--{ !* candidate shared default }--[  ]--
# tools system configuration generate-checkpoint
/system:
    Generated checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-
0.json' with name 'checkpoint-2020-10-20T23:14:12.703Z' and comment ''
```

Example

You can optionally configure a name or comment for a checkpoint; for example:

```
--{ !* candidate shared default }--[  ]--
# save checkpoint comment My-checkpoint
/system:
    Generated checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-
0.json' with name 'checkpoint-2020-10-20T23:23:25.891Z' and comment 'My-checkpoint'
```

Example

The following example configures the system to generate a checkpoint automatically whenever a configuration is committed:

```
--{ !* candidate shared default }--[  ]--
# info system configuration
    system {
        configuration {
            auto-checkpoint true
            }
        }
    }
```

For automatically generated checkpoints, the comment is set to `automatic checkpoint after commit` <*n*>, where <*n*> is the ID of the commit that triggered the checkpoint.

4.13.2 Loading a checkpoint

Procedure

Loading a checkpoint requires an interactive configuration session with a candidate session already active.

Example:

The following example loads a checkpoint into the candidate configuration. Note that you must be in candidate mode to load a checkpoint.

```
--{ * running }--[  ]--
# enter candidate
--{ * candidate shared default }--[  ]--
# tools system configuration checkpoint 0 load
/system/configuration/checkpoint[id=0]:
    Loaded checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-0.json'
```

4.13.3 Reverting to a previous checkpoint

Procedure

You can revert the running configuration to a previous checkpoint. When used within a candidate session, the revert operation loads the checkpoint, removing any present changes, then commits them and establishes a new candidate session.

Example:

The following example reverts the running configuration to a previously saved checkpoint:

```
--{ * running }--[  ]--
# enter candidate
--{ * candidate shared default }--[  ]--
# tools system configuration checkpoint 0 revert
/system/configuration/checkpoint[id=0]:
    Reverting to checkpoint 0 (name 'checkpoint-2019-10-14T18:47:30.282Z'
comment 'Daily_checkpoint')
/:
    Successfully reverted configuration
```

4.13.4 Clearing a checkpoint

Procedure

Checkpoints can be cleared from the system either manually, with a **tools** command, or automatically when the number of saved checkpoints exceeds the configured maximum.

When the number of saved checkpoints exceeds the configured maximum, the oldest checkpoint is removed, and the number of each remaining checkpoint is incremented by 1. If you clear a checkpoint manually, the other checkpoints are not renumbered.

Example:

The following example clears a previously saved checkpoint.

```
# tools system configuration checkpoint 2 clear
/system/configuration/checkpoint[id=2]:
    Cleared checkpoint '/etc/opt/srlinux/checkpoint/checkpoint-2.json'
```

4.13.5 Configuring maximum number of checkpoints

Procedure

By default, the 10 most recent checkpoints are saved; you can configure the maximum number of checkpoint files that are kept by the system.

Example:

The following example configures the system to keep a maximum of 15 checkpoint files.

```
--{ * candidate shared default }--[  ]--
# info system configuration
 system {
     configuration {
         max-checkpoints 15
         }
     }
 }
```

In this example, if 15 checkpoint files are being kept, adding a subsequent checkpoint file causes the oldest checkpoint file to be deleted and the index for the remaining checkpoint files to be incremented by 1.

4.13.6 Displaying checkpoint information

Procedure

Use the **info from state** command to display information about existing checkpoints.

Example:

```
# info from state system configuration checkpoint 0
    system {
        configuration {
            checkpoint 0 {
                name checkpoint-2020-10-20T23:23:25.891Z
                comment cp002
                created 2020-10-20T23:23:25.894Z
                version v20.6.0
                username srlinux
                size 28494
            }
        }
    }
```

4.14 Rescue configuration

You can save a secondary rescue configuration to load in place of the default JSON configuration file. The rescue configuration is a checkpoint file that is loaded automatically by the management server if the default `config.json` fails when the system starts.

If both the default configuration and rescue configuration files are missing or fail, a `config.json` is regenerated and committed from the factory `config.json` that is compiled in the management server.

4.14.1 Saving a rescue configuration

Procedure

You can save a rescue configuration to be loaded automatically if the default `config.json` fails when the system starts. Save the rescue configuration to a file with a **tools** command or using the **save rescue** command. Use these commands in running mode. The system generates a `rescue-config.json` file and saves it to the `/etc/opt/srlinux/checkpoint` directory.

Example:

The following **tools** command saves a rescue configuration:

```
--{ running }--[ ]--
# tools system configuration rescue-save
/system:
    Saved current running configuration as rescue configuration '/etc/opt/srlinux/
checkpoint/rescue-config.json'
```

Example

The following example also saves a rescue configuration:

```
--{ running }--[ ]--
# save rescue
/system:
  Saved current running configuration as rescue configuration '/etc/opt/srlinux/
checkpoint/rescue-config.json'
```

Example

You can confirm the rescue configuration is saved by viewing the checkpoint directory. The following example lists the checkpoint directory:

```
--{ running }--[ ]--
# file ls /etc/opt/srlinux/checkpoint
checkpoint-0.json
rescue-config.json
```

4.14.2 Clearing a rescue configuration

Procedure

To remove an existing rescue configuration, use the **rescue-clear** command to clear the configuration from the `/etc/opt/srlinux/checkpoint` directory. Use this command in running mode. You can then save a new rescue configuration to replace the cleared configuration.

Example:

The following **tools** command clears a previously saved rescue configuration:

```
--{ running }--[ ]--
# tools system configuration rescue-clear
/system:
    Cleared rescue configuration '/etc/opt/srlinux/checkpoint/rescue-config.json'
```

Example

You can confirm the rescue configuration is cleared by viewing the checkpoint directory. The following example lists the checkpoint directory:

```
--{ running }--[ ]--
# file ls /etc/opt/srlinux/checkpoint
checkpoint-0.json
```

4.15 Configuration upgrades

When the SR Linux is started following a software image upgrade, it reads the configuration in the startup `config.json` file, makes any necessary changes to ensure compatibility with the new software image, and places the upgraded configuration into the running configuration. This upgraded configuration is not saved automatically; to save the contents of the running configuration, use the following commands:

- **save startup** – Saves the running configuration to the startup configuration file, located at `/etc/opt/srlinux/config.json` (or `config.gz`).
- **save rescue** – Saves the running configuration to the rescue configuration file, located at `/etc/opt/srlinux/checkpoint/rescue-config.json` (or `rescue-config.gz`).
- **save checkpoint** – Saves the running configuration to a configuration checkpoint; for example, `/etc/opt/srlinux/checkpoint/checkpoint-0.json` (or `checkpoint-0.gz`).
- **save file <name> from running** – Saves the running configuration to the specified file in JSON format.

4.15.1 Upgrading configuration files

Procedure

In addition to saving the upgraded running configuration to startup, rescue, and checkpoint configurations, you can use the following **tools** commands to upgrade existing configuration files so they are compatible with the current software version.

Example:

To upgrade the startup configuration file:

```
--{ running }--[ ]--
# tools system configuration upgrade startup
```

Example

To upgrade the rescue configuration file:

```
--{ running }--[ ]--
# tools system configuration upgrade rescue
```

Example

To upgrade a configuration checkpoint file:

```
--{ running }--[ ]--
```

```
# tools system configuration upgrade checkpoint 0
```

Example

To upgrade a specific JSON-formatted configuration file:

```
--{ running }--[ ]--
# tools system configuration upgrade file /etc/opt/srlinux/configs/myconfig.json
```

5 Securing access

The SR Linux is able to secure access to the device for users connecting via SSH or the console port, as well as for applications and FTP access. Authentication can be performed for users configured within the underlying Linux OS, and for administrative users configured within the SR Linux itself.

Depending on the user type, users are authenticated locally on the device or through interaction with the SR Linux aaa_mgr application and an authentication server group (for example, TACACS+).

5.1 User types

The SR Linux supports three user types: Linux users, local users, and remote users. Each user type is authenticated differently, as described in the following sections.

5.1.1 Linux users

Linux users are those configured in the underlying Linux OS, not in the SR Linux CLI. Information about Linux users is stored in `/etc/passwd` in the underlying Linux OS.

By default, the SR Linux has a single Linux user, `linuxadmin`, who has access to `sudo` to root and can run the SR Linux CLI with admin permissions. The default password for the `linuxadmin` user is `Nokia Srl1!`, which you can change using the SR Linux CLI; see Configuring the password for the linuxadmin user. Nokia recommends that you change this default password as soon as possible. Other Linux users can be added with the `useradd` command in the underlying Linux OS.

Linux users, as long as they have a UID less than 1500, are authenticated via the underlying Linux OS, not through the SR Linux aaa_mgr application. This means that Linux users are not subject to authentication settings configured within the SR Linux CLI, such as authentication by a TACACS+/RADIUS server group or password security for local users.

5.1.2 Local users

Local users are users configured within the SR Linux itself. By default, SR Linux supports a single local user, named `admin`; other local users can be added as necessary.

Local users are authenticated via a gRPC interface to the aaa_mgr application. See Authentication for local users for an example configuration.

5.1.3 Remote users

Remote users are users that are not configured either in `/etc/passwd` or within the SR Linux configuration. Remote users are configured on a remote server, which is queried when the user attempts to log in to the SR Linux device.

5.2 AAA functions

The SR Linux performs authentication, authorization, and accounting (AAA) functions for each user type, as described in the following sections.

5.2.1 Authentication

For Linux users, the SR Linux authenticates via the authentication mechanism built into the underlying Linux OS.

For local users, including the SR Linux `admin` user, the SR Linux uses its gRPC interface to the aaa_mgr application for authentication. Authentication settings that apply to the local users, including a local password and TACACS+ server group, can be configured using the SR Linux CLI. See Authentication for local users for information about configuring local users.

For remote users, authentication is performed using the aaa_mgr application in coordination with the remote system.

5.2.1.1 Password security for local users

SR Linux supports password security features that apply to local users who authenticate using plain text passwords. The settings are system-wide configurations that apply to all locally configured users including the admin user, but excluding the linuxadmin user. The following password security features are supported:

- **Password complexity rules**

  Specifies rules that define the password requirements, including minimum and maximum length, minimum lowercase and uppercase characters, minimum numeric and special characters, and whether the password can include the user's full username or only the first three characters. The configuration of password complexity rules applies only to new passwords entered as clear text, not to pre-existing or pre-hashed ones.

- **Password aging**

  Specifies the number of days after which a password will expire.

  Note:

  – Users with aged out passwords are prompted to update their password at the next login to the CLI. (This prompt appears on the CLI only.)

  – SR Linux does not provide advanced warning to users that their password is about to expire.

  – Aged out passwords expire indefinitely. They are not controlled by the lockout policy described below.

- **Password change on first login**

  Specifies whether local users must change their password on first login. The new password must match the defined password complexity rules.

- **Password history**

  Specifies the number of previous user passwords SR Linux retains to prevent reuse of old passwords.

- **Lockout policy**

  Specifies how many times a user can attempt a failed login within a defined period before they are locked out (for example: three attempts within 1 minute). The policy also defines whether the user is locked out indefinitely or only for a specified length of time.

5.2.2 Authorization

The SR Linux implements authorization through role-based access control, where each authenticated user is assigned one or more predefined roles that specify the functions the user is allowed to perform on the system. If no role is configured for a user, then the user is assigned a role that allows access to CLI plugins, but no other functions.

Authorization via role based access control is performed for all user types on all interfaces (CLI, gNMI, JSON-RPC), with the exception of the `admin` and `linuxadmin` users, which are permitted write access to all commands in the command tree. You can configure service authorization, which can limit the interface types for which the functions in a role are authorized.

You can configure the SR Linux to use information from a TACACS+ server to assign roles to an authenticated user. In this case, the priv-lvl value in the authorization-reply message received from the TACACS+ server maps to a role configured on the SR Linux. The user is assigned the role that corresponds to the priv-lvl value.

See Authorization using role-based access control for information about configuring roles for users.

5.2.3 Accounting

The SR Linux supports command accounting. Accounting records generated by the SR Linux include the entire CLI string that a user enters on the command line, including any pipes or output redirects specified in the command.

You can configure the SR Linux device to send accounting records to a destination specified in an accounting-method list, such as a TACACS+ server group or the local system.

For each user type, the SR Linux device generates accounting records as follows:

- For local users, including the SR Linux `admin` user, command accounting records are sent to the destination specified in the accounting-method list, both for commands entered in the SR Linux CLI and for commands entered in the bash shell.
- For Linux users and remote users, command accounting records are sent for commands entered in the SR Linux CLI (including Linux commands entered in the SR Linux CLI using the **bash** command), although not for commands entered in the bash shell.

See Configuring accounting for an example configuration.

5.3 AAA server group configuration

The SR Linux supports the following server group types for AAA functions:

- local – Uses local authentication, including `/etc/passwd`, `/etc/group`, and logging via syslog.
- TACACS+ – Performs authentication, authorization, and accounting via interaction with servers in a TACACS+ server group.

Users whose AAA functions are handled by the aaa_mgr application (that is, the SR Linux `admin` user and local users) can use one of these server groups for authentication and accounting.

The TACACS+ server group can have up to five servers. When authenticating a user or writing an accounting record, the SR Linux tries each server in the group in a round-robin fashion until a response is received. If no response is received within a specified timeout period, the SR Linux tries the next server in the group.

If no response is received from any of the servers in the group, the SR Linux moves to the next specified authentication or accounting method. If no other method is specified, or the TACACS+ server group is the last method in the list, then the authentication or accounting request is rejected.

5.3.1 Configuring an AAA server group

Procedure

The following example shows settings for a TACACS+ and a local server group to be used for AAA functions. TACACS+ requests are sourced from the mgmt network-instance.

The TACACS+ server group consists of three TACACS+ servers. The timeout period specifies that the SR Linux wait 30 seconds for a response from a server before trying the next server in the group.

For the server group of type local, no external servers can be specified. The local server group uses `/etc/passwd` and `/etc/group` for authentication, and syslog for accounting. The timeout period specifies that the SR Linux wait a maximum of 60 seconds for an AAA function to complete.

Example:

```
--{ * candidate shared default }--[  ]--
# info system aaa
 system {
     aaa {
         server-group tacacs-all {
            type tacacs
            timeout 30
            server 1.2.2.1 {
                network-instance mgmt
                tacacs {
                    secret-key $aes$3/Iz5veTDRV0=$6GxkrGjFbqWbYMA0T3itgQ==
                }
            }
            server 1.2.2.2 {
                network-instance mgmt
                tacacs {
                    secret-key $aes$3/Iz5veTDRV0=$6GxkrGjFbqWbYMA0T3itgQ==
                }
            }
            server 1.2.2.3 {
                network-instance mgmt
                tacacs {
                    secret-key $aes$3/Iz5veTDRV0=$6GxkrGjFbqWbYMA0T3itgQ==
                }
            }
         server-group local {
            type local
            timeout 60
            }
         }
     }
```

```
}
```

5.4 Authentication for the linuxadmin user

The SR Linux `linuxadmin` user is installed by default in `/etc/passwd`. The `linuxadmin` user has access to `sudo` to root and can run the SR Linux CLI with admin permissions.

The default password for the `linuxadmin` user is `NokiaSrl1!`. You can change this password via the SR Linux CLI or any other supported interface.

5.4.1 Configuring the password for the linuxadmin user

Procedure

Use the following command to set the password for the `linuxadmin` user.

Example

```
--{ * candidate shared default }--[  ]--
# system aaa authentication linuxadmin-user password NewPass1234
```

When the user configuration is displayed, the password is hashed; for example:

```
--{ * candidate shared default }--[  ]--
# info system aaa authentication linuxadmin-user
    system {
        aaa {
            authentication {
                linuxadmin-user {
                    password $6$d091a49828c4cc87$CGe7QbxzZm/WadeMYF
                }
            }
        }
    }
```

5.5 Authentication for local users

Local users are those configured within the SR Linux CLI. For a local user, you can configure a password and specify one or more authentication methods, including local authentication or a TACACS+ server group.

5.5.1 Configuring authentication for local users

Procedure

By default, there is a single local user configured on the SR Linux, `admin`. You can configure additional local users.

Example:

The following example configures a password for the SR Linux `admin` user:

```
--{ * candidate shared default }--[  ]--
# system aaa authentication admin-user password NewPass1234
```

Example

The following example creates a local user called `sruser` and configures a password:

```
--{ * candidate shared default }--[  ]--
# system aaa authentication user srlinux password sr1l234
```

When the user configuration is displayed, the password is hashed; for example:

```
--{ * candidate shared default }--[  ]--
# info system aaa
    system {
        aaa {
            authentication {
                user srlinux {
                    password $ar2$KGSITqfJu5g=$iZZ6XKXtXOZOGMbeXO2ypg==
                }
            }
        }
    }
```

Example

To change an existing user's password, use the same command that created the user and configure the new password; for example:

```
--{ * candidate shared default }--[  ]--
# system aaa authentication user srlinux password NewPasswordl234
```

Example

The following example specifies authentication methods. When a user attempts to log in, the user is authenticated using local authentication first. If local authentication fails, the SR Linux tries the servers in TACACS+ server group tacacs-all. If the user cannot be authenticated through either method, the authentication attempt is rejected.

```
--{ * candidate shared default }--[  ]--
# info system aaa
 system {
     aaa {
         authentication {
            authentication-method [
                local
                tacacs-all
            ]
         }
     }
 }
```

5.5.2 Configuring password security for local users

Procedure

Password security and user lockout settings apply to all local users who authenticate using plain text passwords including the admin user.

To configure password security and user lockout features, set the following parameters under **system aaa authentication**:

- **password aging** *<0 to 500>*
  Sets the number of days after which the user password expires and the user is prompted to update their password. The default of **0** sets the password to never expire.
- **password change-on-first-login [true | false]**
  When set to **true**, forces local users to change their password on the first login to the system. The default is **false**.
- **password history** *<0 to 20>*
  Specifies how many previous passwords SR Linux matches a new password against, such that the new password cannot be one of the previous *<0 to 20>* passwords. The default is **0**, which disables password history.
- **password complexity-rules**
  Sets the complexity rules that new passwords must match, using the following options:
  - **minimum-length** *<1 to 12>*
    Sets the minimum password length. The default is **1**, which applies no minimum.
  - **maximum-length** *<1 to 1023>*.
    Sets the maximum password length. The default is **1**, which applies no maximum.
  - **minimum-lowercase** *<0 to 10>*
    Sets the minimum required lowercase characters. The default is **0**, which applies no minimum.
  - **minimum-uppercase** *<0 to 10>*
    Sets the minimum required uppercase characters. The default is **0**, which applies no minimum.
  - **minimum-numeric** *<0 to 10>*
    Sets the minimum required numeric characters. The default is **0**, which applies no minimum.
  - **minimum-special-character** *<0 to 10>*
    Sets the minimum required special characters: (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"). The default is **0**, which applies no minimum.

    Note: Users must enter the double quote as \" and the backslash as \\.

  - **allow-user-name** [**true** | **false**]
    Specifies whether the user can include their username as part of their password. If set to **false**, then SR Linux allows only the first three consecutive characters of the username in the user password. The default is **true**, which allows the password to contain the full username.
- **password lockout-policy**
  Specifies the user lockout policy using the following parameters:
  - **attempts** *<0 to 64>*
    Sets the number of failed login attempts that trigger the lockout. The default is **0**, which allows unlimited failed login attempts.

- **time** *<0 to 1440>*
  Sets the time period in minutes within which the failed login attempts must occur to trigger the lockout. The timer starts at the first failure. The default is **1** minute.
- **lockout** *<0 to 1440>*
  Sets the time period in minutes during which the user account remains locked out. A value of **0** means that the user account is locked out indefinitely. The default is **15** minutes.

Example: Configure password security and user lockout

The following example sets the following password security and user lockout settings:

- User passwords expire after 365 days.
- Users must change their password on first login.
- Users cannot reuse any of their 10 previous passwords.
- Passwords must contain:
  - A minimum of eight and a maximum of 16 characters.
  - One each of lowercase, uppercase, numeric, and special characters.
- Usernames are not allowed in the passwords.
- After five failed attempts within 2 minutes, users are locked out indefinitely.

```
--{ * candidate shared default }--[ ]--
# info aaa authentication password
    aaa {
        authentication {
            password {
                aging 365
                change-on-first-login true
                history 10
                complexity-rules {
                    minimum-length 8
                    maximum-length 16
                    minimum-lowercase 1
                    minimum-uppercase 1
                    minimum-numeric 1
                    minimum-special-character 1
                    allow-username false
                }
                lockout-policy {
                    attempts 5
                    time 2
                    lockout 0
                }
            }
        }
    }
```

5.5.3 Clear local user lockout

Procedure

To display locked out users, use the **info from state system aaa authentication** command, which displays a `lockout` state of `active true` when a user is locked out.

To clear a local user's lockout state, use the following tools command:

tools system aaa authentication user <*username*> **unlock**

Example: Display user lockout state

```
--{ * candidate shared default }--[ ]--
#info from state system aaa authentication
    system {
        aaa {
            authentication {
                exit-on-reject false
                idle-timeout 600
                authentication-method [
                    local
                ]
                user srl-test-1 {
                    password $bt$Vbf0Zgg8MGP=$ihHklTQHi/+3VwVD04Z8gh==
                    failed-login-attempts 10
                    last-failed-login 2022-07-29T22:35:23.801Z
                    password-change-required true
                    role [
                        test
                    ]
                    lockout {
                        active true
                        start 2022-07-29T22:35:23.801Z
                        end "14 minutes from now"
                    }
                }
                user srl-test-2 {
                    password $bt2$jBZBL6WAuTr=$eXYWpivE5z10YxBIZ06hjQ==
                    password-change-required true
                    role [
                        test
                    ]
                }
```

Example: Clear user lockout state

```
--{ * candidate shared default }--[ ]--
#tools system aaa authentication user srl-test-1 unlock
/system/aaa/authentication/user[username=srl-test-1]:
    Unlocked user srl-test-1
```

5.6 Authorization using role-based access control

Authorization is performed through role-based access control. Users can be configured with a set of one or more roles that indicate the privileges for which they are authorized in the system.

A role consists of one or more rules, which specify a schema path the role can have privileges for, and a corresponding action, which can be read, write, or deny. After authentication, a user is authorized to perform the specified action defined in the path for the role the user is assigned.

Role-based access control supports service authorization, which allows you to limit the actions a user is authorized to perform to specific access types such as CLI and gNMI.

5.6.1 Role configuration

Roles consist of a set of rules that define a schema path-reference and a corresponding action. The path-reference specifies system functions that are subject to authorization, and the action specifies the privilege type for users assigned the role: read, write, or deny.

The path-reference is specified relative to the root level. For example, the path-reference "`/`" indicates all CLI functions at the root level and below; the path-reference "`/system`" indicates all CLI functions at the `system` level and below, and the path-reference "/system configuration" indicates all CLI functions at the `system configuration` level and below.

If no role is assigned to a user at login, the user has access to all CLI plugins, but no access to CLI commands at any level. This is equivalent to a rule with `/` as the path-reference and `deny` as the action.

Up to 32 roles are supported; each role can have a list of up to 256 paths. The order of the path-references in a role does not matter; the longest match is used when validating a command against a path-reference.

The syntax for the path-reference is SR Linux CLI format, with "`/`" representing the root. Wildcards and ranges can be used with path-references, in the same way they can in CLI syntax.

Each entry within the path-reference should use double quotes, unless the command string is a single word with no spaces. If the path itself includes quotes, use backslash characters to indicate the quotes. For example, to specify the following in a path-reference:

`a "b"` and `c "d"`

You can configure the following for the path-reference:

path-reference `"a \"b\"" "c \"d\""`

5.6.1.1 Configuring a role

Procedure

To configure a role, you define one or more rules that specify a path indicating the command string that is subject to authorization, and a corresponding action such as read, write, or deny.

Example:

The following is an example of creating a role named `testrole` that contains two rules: one rule to limit access to network-instance `red`, and another rule to give write access to the rest of the tree. A user assigned this role upon authentication is able to configure everything in the system except network-instance `red`.

First, define the role under the `system aaa authorization` context:

```
--{ * candidate shared default }--[  ]--
# info system aaa
    system {
        aaa {
            authorization {
                role testrole {
                }
            }
        }
    }
```

Then specify the rules under the `system configuration role` context:

```
--{ * candidate shared default }--[  ]--
# info system configuration
    system {
        configuration {
            role testrole {
                rule / {
                    action write
                }
                rule "network-instance red" {
                    action read
                }
            }
        }
    }
```

Example

The following example configures a role named `acl_app` that allows a user to view the state of the acl_mgr application, but no other information:

```
--{ * candidate shared default }--[  ]--
# info system aaa
    system {
        aaa {
            authorization {
                role acl_app {
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# info system configuration
    system {
        configuration {
            role acl_app {
                rule system {
                    action deny
                }
                rule "system app-management application acl_mgr state" {
                    action read
                }
            }
        }
    }
```

When a user that is assigned the `acl_app` role attempts to read information from the system state datastore, only the state of the acl_mgr application is displayed. For example:

```
--{ * candidate shared default }--[  ]--
# info from state system application acl_mgr
system {
    app-management {
        application acl_mgr {
            state running
        }
    }
}
```

5.6.2 Assigning roles to users

Procedure

A user can be assigned one or more roles. The rules configured in the user's roles specify the commands the user is authorized to issue.

Up to 32 roles can be assigned to a user. If a user has multiple roles assigned, all of the rules configured in all of the roles apply to the user. If multiple roles reference the same path, the most specific rule is used.

For service authorization, the system merges all roles that have the service (CLI, gNMI, and so on) the user has logged in with, and excludes any roles that omit the service.

Example:

In the following example, the user `srlinux` is assigned the role `testrole`.

After the `srlinux` user is authenticated, the user is authorized to use the system according to the rules configured in the role `testrole`. Using the example from Configuring a role, this would authorize the `srlinux` user to configure everything in the system except for the network-instance `red`.

```
--{ * candidate shared default }--[  ]--
# info system aaa
    system {
        aaa {
            authentication {
                user srlinux {
                    role [
                        testrole
                    ]
                }
            }
        }
    }
```

5.6.3 Authorization using a TACACS+ server

You can configure the SR Linux to use a TACACS+ server to provide authorization for role-based access control. When TACACS+ authorization is configured, the actions an authenticated user can perform depend on the priv-lvl value configured for the user on a TACACS+ server. If **priv-lvl-authorization** is not set to **true**, the authenticated user is authorized as an admin user.

TACACS+ authorization for SR Linux users works as follows:

1. After a user is authenticated, the SR Linux sends the TACACS+ server an authorization-request message based on a shell (exec) session starting. This authorization-request is sent immediately following user authentication, but before the shell is started.
2. The TACACS+ server returns an authorization-reply message that includes the priv-lvl value configured for the user for shell access.
3. On the SR Linux, roles have been configured that map to a priv-lvl value.
4. The SR Linux allows the user to perform actions specified in roles with a priv-lvl value equal or lower to the priv-lvl value returned by the TACACS+ server.

5.6.3.1 Configuring TACACS+ Authorization

Procedure

To configure TACACS+ authorization for SR Linux users, you enable priv-lvl authorization for the TACACS + server group and configure roles that specify the priv-lvl mapping for each role subject to authorization using a TACACS+ server.

Example:

The following configuration enables priv-lvl authorization for the `tacacs-all` server group:

```
--{ * candidate shared default }--[  ]--
# info system aaa server-group tacacs-all
    system {
        aaa {
            server-group tacacs-all {
                priv-lvl-authorization true
            }
        }
    }
```

Example

The following configuration specifies two roles, `interface oper-state` and `network-instance oper-state`. The `interface oper-state` role grants read access for all interface oper-states, and the `network-instance oper-state` role grants read access for all network-instance oper-states.

```
--{ * candidate shared default }--[  ]--
# info system configuration role *
    system {
        configuration {
            role "interface oper-state" {
                rule "interface * oper-state" {
                    action read
                }
            }
            role "network-instance oper-state" {
                rule "network-instance * oper-state" {
                    action read
                }
            }
        }
```

Example

The following configuration specifies the priv-lvl mapping for both roles.

In this example, when a user is authenticated, the SR Linux sends an authorization-request based on a shell (exec) session starting. The TACACS+ server returns an authorization-reply that specifies the priv-lvl value for the shell.

If the priv-lvl value returned by the TACACS+ server is 14, the user is assigned both roles; that is, read access to all interface oper-states and all network-instance oper-states, but no access to anything else in the system.

If the priv-lvl value returned by the TACACS+ server is 13, the user is assigned only the `network-instance oper-state` role and can read the oper-state for network-instances, but has no access to anything else in the system.

```
--{ * candidate shared default }--[  ]--
# info system aaa authorization
    system {
        aaa {
            authorization {
                role "interface oper-state" {
                    tacacs {
                        priv-lvl 14
                    }
                }
                role "network-instance oper-state" {
                    tacacs {
                        priv-lvl 13
                    }
                }
            }
        }
    }
```

Example

The following example configures service authorization for a role that uses TACACS+ authorization. See Configuring service authorization.

In this example, when a user is authenticated by a TACACS+ server in the server group `tacacs-all`, if the TACACS server returns priv-lvl 15 for the user, then `role_1` is assigned to the user.

Service authorization is configured for the role so that all services except gNMI are authorized for users assigned this role. This means users assigned `role_1` are authorized for the functionality defined in the rules of `role_1` if they connect using the CLI, JSON-RPC, or FTP, but are not authenticated if they connect using gNMI.

```
--{ * candidate shared default }--[  ]--
# info system aaa authorization role role_1
    system {
        aaa {
            authorization {
                role role_1 {
                    services [
                        cli
                        json-rpc
                        ftp
                    ]
                    tacacs {
                        priv-lvl 15
                    }
                }
            }
        }
    }
```

When a user is assigned multiple roles, the user is authorized for all services specified in the roles they are assigned, according to the rules defined in the roles. For example, if a user is assigned `role_1`, which allows access to the system via CLI, and also assigned `role_2`, which allows access via gNMI, the user is authorized to use the CLI to perform the functions defined in the rules of `role_1`, and is authorized to use the gNMI interface for the functions defined in the rules of `role_2`.

5.6.4 Service authorization for local users

Service authorization allows you to block or allow access to a user depending on the interface they use to connect to the device. You can use service authorization to restrict access to a controller, allowing it to speak through programmatic interfaces, but without credentials that can be used by someone logged into the CLI.

To configure service authorization, you assign roles to local users that authorize them to issue commands using specified services, such as the CLI, gNMI, or JSON-RPC interfaces. For example, you can define a role that grants read access to subinterface statistics, and limits access to that information to the gNMI service. When you assign that role to a user, the user is allowed to read subinterface information via the gNMI interface only.

The following interface types can be configured for service authorization:

- `cli` - access to the system via CLI
- `gnmi` - access to the system via gNMI
- `json-rpc` - access to the system via JSON-RPC
- `ftp` - access to the system via FTP
- `p4rt` - access to the system via P4Runtime RPCs

After a user is authenticated, the system checks whether the user is assigned a role that allows access via the interface they used to connect to the device. For example, if a user connects using gNMI, the system checks whether the user is assigned any roles that authorize access via the gNMI interface. If the user is assigned a role that authorizes access via gNMI, the user receives access to the system via gNMI according to the rules defined in the set of roles that includes gNMI. If the user connects via gNMI, but is not assigned any role that authorizes access via gNMI, the authorization attempt is rejected and the session is closed.

If a user is assigned multiple roles, the user is authorized for all services specified in the roles they are assigned, according to the rules defined in the roles. For example, consider a user assigned role r1, which allows access to the system via CLI, and also assigned role r2, which allows access via gNMI. The user is authorized to use the CLI to perform the functions defined in the rules of role r1, and is authorized to use the gNMI interface for the functions defined in the rules of role r2.

By default, a role has no services configured for authorization. When you configure a role, you must specify the services that apply to the role. Users assigned the role are authorized to perform the functions defined in the role using the specified services.

5.6.4.1 Configuring service authorization

Procedure

To configure service authorization, you create a role that defines rules permitting or denying access to system functionality, assign the role to one or more users, and specify the services over which users assigned the role are authorized to perform the rules defined in the role.

Example:

The following example creates a role `read-oper-state` that allows reading subinterface oper-state but nothing else:

```
--{ * candidate shared default }--[  ]--
# info system configuration role *
    system {
        configuration {
            role read-oper-state {
                rule / {
                    action deny
                }
                rule "interface * subinterface * oper-state" {
                    action read
                }
            }
        }
    }
```

Example

The following example creates a user `gnmiuser` that is assigned the `read-oper-state` role when authenticated:

```
--{ * candidate shared default }--[  ]--
# info system aaa authentication
    system {
        aaa {
            authentication {
                authentication-method [
                    local
                ]
                user gnmiuser {
                    role [
                        read-oper-state
                    ]
                }
            }
        }
    }
```

Example

The following example configures service authorization so that users assigned the `read-oper-state` role, such as `gnmiuser`, can perform the functionality defined in the role only if they have connected to the system via gNMI. If the user connects via a different service, such as by logging into the CLI, the user is not authorized for the functionality defined in the role.

```
--{ * candidate shared default }--[  ]--
# info system aaa authorization
    system {
        aaa {
            authorization {
                role read-oper-state {
                    services [
                        gnmi
                    ]
                }
            }
        }
```

```
    }
```

5.7 Accounting configuration

When accounting is enabled, the SR Linux device generates command accounting records as described in Accounting.

The following is an example of accounting records generated by the SR Linux device:

```
Aug 7 22:34:09
127.0.0.1 bob ssh 172.17.0.1 start task_id=2 timezone=UTC service=shell priv-lvl=15
cmd=tail -f /var/log/tac_plus.acct
Aug 7 22:34:09
127.0.0.1 bob ssh 172.17.0.1 stop task_id=2 timezone=UTC service=shell priv-lvl=15
cmd=tail -f /var/log/tac_plus.acct
Aug 7 22:34:14
127.0.0.1 bob ssh 172.17.0.1 start task_id=5 timezone=UTC service=shell priv-lvl=15
cmd=help
Aug 7 22:34:14
127.0.0.1 bob ssh 172.17.0.1 stop task_id=5 timezone=UTC service=shell priv-lvl=15
cmd=help
```

5.7.1 Configuring accounting

Procedure

The following example configures accounting records to be sent to the `tacacs-all` server group. The SR Linux generates an accounting record when a command is started and when it is stopped.

Example:

```
--{ * candidate shared default }--[  ]--
# info system aaa
 system {
     aaa {
         accounting {
            accounting-method [
                tacacs-all
            ]
            event commands {
                record start-stop
            }
         }
     }
 }
```

5.8 Displaying user session information

Procedure

To display information about users currently logged in to the SR Linux device, use the **show system aaa authentication session** command.

Example:

```
# show system aaa authentication session
+----+-------+-------------+----------------+----------+-------+-------------+----------------------+
| ID | User  |   Service   | Authentication | Priv-lvl |  TTY  | Remote  |      Login time         |
|    | name  |    name     |     method     |          |       |  host   |                         |
+====+=======+=============+================+==========+=======+=========+=========================+
| 4  | bob   | srlinux-cli |    tacacs      |    15    | pts/1 | 2.1.0.2 | 2021-12-06T21:24:07.80Z |
| 11 | user* | srlinux-cli |    local       |          | pts/4 |         | 2021-12-07T04:06:06.93Z |
+----+-------+-------------+----------------+----------+-------+---------+-------------------------+
```

The asterisk (*) next to the username indicates the session from which the **show system aaa authentication session** was invoked.

5.9 Disconnecting user sessions

Procedure

To disconnect a user currently logged in to the SR Linux device, use the **tools system disconnect session-id <session-id>** command and specify the session ID of the user. To list the session IDs of active users, enter the **show system aaa authentication session** command.

Example:

```
# show system aaa authentication session
+----+-------+-------------+----------------+----------+-------+-------------+----------------------+
| ID | User  |   Service   | Authentication | Priv-lvl |  TTY  | Remote  |      Login time         |
|    | name  |    name     |     method     |          |       |  host   |                         |
+====+=======+=============+================+==========+=======+=========+=========================+
| 6  | bob   | srlinux-cli |    tacacs      |    15    | pts/1 | 2.1.0.2 | 2021-12-06T21:24:07.80Z |
| 11 | user* | srlinux-cli |    local       |          | pts/4 |         | 2021-12-07T04:06:06.93Z |
+----+-------+-------------+----------------+----------+-------+---------+-------------------------+

# tools system disconnect session-id 6
Terminating cli session 6 owned by user 'bob' logged from pts/1
/system/aaa/authentication/session[id=6]:
    Disconnecting aaa cli session(s): 6
```

5.10 Configuring idle-timeout for user sessions

Procedure

You can configure the idle-timeout for user sessions, which disconnects a user session after a specified period of inactivity. By default, user sessions are disconnected after 15 minutes of inactivity.

The idle-timeout setting applies to SR Linux users and remote users. It does not apply to Linux users or to JSON-RPC or gNMI client sessions.

After a user session has been inactive for one-half of the idle-timeout period, a notification is displayed indicating that the user will be logged out if the session remains idle for the remainder of the idle-timeout period.

Example:

The following example configures the idle-timeout so that SR Linux user sessions and remote user sessions are disconnected after 20 minutes of inactivity:

```
--{ * candidate shared default }--[  ]--
# info system aaa
 system {
     aaa {
         authentication {
            idle-timeout 20
            }
         }
     }
 }
```

6 Management servers

You can configure the following management servers on the SR Linux:

- gNMI server – allows external gNMI clients to connect to the device and modify the configuration and collect state information.
- JSON-RPC server – allows you to issue JSON-formatted requests to the device to retrieve and set configuration and state.

You can configure Transport Layer Security (TLS) profiles, which contain TLS settings that can be provided to the gNMI and JSON-RPC management servers.

6.1 gNMI server

The SR Linux device can enable a gNMI server that allows external gNMI clients to connect to the device and modify the configuration and collect state information.

When the gNMI server is enabled, the SR Linux gnmi_mgr application functions as a target for gNMI clients. The gnmi_mgr application validates gNMI clients and passes Get, Set, and Subscribe RPCs to the SR Linux mgmt_svr application via the gRPC interface. See the *SR Linux System Management Guide* for details about the supported RPCs.

Configuration changes made by gNMI clients are made within a private candidate configuration, using a snapshot of the current running configuration as a baseline for the private candidate. As with other types of candidate configurations, the private candidate can operate in exclusive mode, which locks out other users from concurrently modifying the private candidate configuration.

Sessions between gNMI clients and the SR Linux device must be encrypted using TLS. You can specify TLS settings within a TLS profile, and apply the TLS profile when configuring a gNMI server within a network-instance. When the gNMI server is enabled, gNMI clients connect and authenticate to the SR Linux device using the settings specified in the TLS profile.

New connections between gNMI clients and the SR Linux device are mutually authenticated. By default, the SR Linux device validates the X.509 certificate of the gNMI client, and the other way around; this behavior can be disabled in the TLS profile. The SR Linux device, after validating the X.509 certificate of the gNMI client, performs local authentication, if the **use-authentication** parameter is set to **true**. In this case, the gNMI client is required to provide a username and password in the metadata of the Get, Set, and Subscribe RPCs. The supplied username and password are authenticated by the SR Linux aaa_mgr application.

See the *SR Linux System Management Guide* for examples of using the gnmi_cli, gnmi_get, and gnmi_set open source gNMI clients to configure and retrieve state information about the SR Linux device.

6.1.1 Configuring a gNMI server

Procedure

The SR Linux supports configuring a gNMI server under one or more network-instances. You can specify limits for the number of simultaneous active gNMI client sessions, as well as the number of connection attempts per minute by gNMI clients.

For the network-instance where the gNMI server is running, you can specify the IP address and port for gNMI client connections, as well as the TLS profile used for authenticating gNMI clients. See TLS profiles.

Example:

The following example shows a configuration that enables a gNMI server on the SR Linux device. The gNMI server is configured so that gNMI clients can connect to the SR Linux via the mgmt network-instance on port 50052. Connecting gNMI clients are authenticated using the settings specified in the TLS profile `tls-profile-1`.

```
--{ * candidate shared default }--[  ]--
# info system gnmi-server
 system {
     gnmi-server {
         admin-state enable
         timeout 7200
         rate-limit 60
         session-limit 20
         network-instance mgmt {
             admin-state enable
             use-authentication true
             port 50052
             tls-profile tls-profile-1
             source-address [
                 ::
             ]
         }
     }
 }
```

6.2 JSON-RPC server

You can enable a JSON-RPC server on the SR Linux device, which allows you to issue JSON-formatted requests to the device to retrieve and set configuration and state. You can use the JSON-RPC API to run CLI commands and standard get and set methods. The SR Linux device returns responses in JSON-format.

Configuration changes made using the JSON-RPC API are made within a private candidate configuration, using a snapshot of the current running configuration as a baseline for the private candidate. As with other types of candidate configurations, the private candidate can operate in exclusive mode, which locks out other users from concurrently modifying the private candidate configuration.

When the JSON-RPC server is enabled, the application passes the requests to the SR Linux mgmt_svr application via the gRPC interface.This JSON-RPC API uses HTTP and HTTPS for transport, and users are authenticated with the aaa_mgr application. HTTPS requests can be authenticated using TLS, using settings specified in a TLS profile. See TLS profiles.

See the *SR Linux System Management Guide* for examples of using the get method to retrieve state information from the SR Linux, the set method to modify the SR Linux configuration, and the cli method to enter SR Linux CLI commands.

6.2.1 Configuring a JSON-RPC server

Procedure

The SR Linux supports configuring a JSON-RPC server under one or more network-instances. You can specify limits for the number of simultaneous active HTTP or HTTPS connections and the TCP port used for HTTP or HTTPS connections. If the TCP port is in use when the JSON-RPC server attempts to bind to it, the commit operation fails.

Example:

The following example shows a configuration that enables a JSON-RPC server within the mgmt network-instance on the SR Linux device. The JSON-RPC server is configured so that HTTP requests are accepted on TCP port 4000, and HTTPS requests are accepted on TCP port 443. HTTPS requests are authenticated using the settings in the TLS profile `tls-profile-1`.

```
--{ * candidate shared default }--[  ]--
# info system json-rpc-server
 system {
     json-rpc-server {
         admin-state enable {
         network-instance mgmt {
             http {
                 admin-state enable
                 use-authentication true
                 session-limit 1
                 port 4000
             }
             https {
                 admin-state enable
                 use-authentication true
                 session-limit 1
                 port 443
                 tls-profile tls-profile-1
             source-address [
                 ::
             ]
             }
     }
 }
```

Note:
To connect to the configured JSON-RPC server, enter the <*source-address*> defined in the **system json-rpc-server** configuration, followed by `/jsonrpc`:

`https(s)://<source-address>/jsonrpc`

6.3 TLS profiles

Transport Layer Security (TLS) is a protocol for enabling applications or devices to exchange information. The SR Linux supports configuring TLS settings in TLS profiles, which can be provided to applications

such as the gNMI server and the JSON-RPC server, so that clients connecting to the SR Linux device via these applications are authenticated using the settings in the TLS profile.

6.3.1 Configuring a TLS profile

Procedure

A TLS profile can specify whether authentication is performed for clients connecting to SR Linux applications to which the profile is applied. Within a TLS profile, you can configure certificates, keys, and ciphers to use when negotiating TLS connections with clients.

Example:

```
--{ * candidate shared default }--[  ]--
# info system tls
 system {
     tls {
         server-profile tls-profile-1 {
                key $aes$4NAaR4Zz5skA=$3Pv773cUer0TaPNHqRQ==
                certificate "-----BEGIN CERTIFICATE REQUEST-----
MIIEUjCCAjoCAQAwDTELMAkGA1UEBhMCVVMwggIiMA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDxG7nZ
qw7EjHM6uBit7Bi1gO/PgQTD2OqJTOtFqEXcFzGcbeeFk903v7OTNws0h11A1pZGnT9fPXv/
hYmAcvOv4FZz99jrGZ8WiwJMpxG+wG3rNdCwEfc59cqFOu9k8pbxLZVYck+pVcjgUK9wLZMHtDVYADXp7I5h
NYMLdettpSSucXaZcWpTzD/xJtePa9dkQ75LGcl/
JMivhx+7EYsbBRlkoSeluMByGavxjefNPJFtvOUAPE7CvdMprntHA7op5FkSoqZcoTzA0V1BcaNtblU7j8DL
1UnsRk6Mi9M5Sz5McQDW8cn223pT9AceLCM27LY62rNEcpAZHNumPqG352+mdLqZ7sJmfwScB3EISj6YV+ni
sZ+K8woR7PD0oz3rsnYGjjtE3xf/
NwXNdioFaVF80nkhwbpoSYZFuwzigkBvsyeUQzmYe4ehC5eFezmWracItmsqzjsDqoJZa5y3ngAK72ag9wQN
2Lz3AHYvMlC3Gn4D2P/
oNH1ywL0DeSdEMxukQamSF8EiEvCu1cBkBhab3blVOp61c7IsaNHdW1k95cFpfNLT+1HoWJlkaIVI7gCTgPW
2UURy67lUBAl4rdpFnnkRLnJfKYgjScLWSw0ur17NOTinflAgx4k5AXZP8FpvW3S0KyqR0S8UHajcsYsRd7W
aIVOhFOeXUxYeGQIDAQABoAAwDQYJKoZIhvcNAQELBQADggIBAFdyqd7yGmbM9E12i4y7nwUvORg5nwr6b5Z
aCGnCl3h8bYerhS/QjNTahAJYKl+G2pYCOdfq435B8NNFdsEfBJ9Z8C6Vs/
kixlfFVGGZVaglxR9Vgs13Srht0aE4LgE2BiUwJe5szicGRr7M/OXCN/
yS2fH5qbuHyJHdP3UiEliNleuYLO+U0ALN0XXVrFJ+FF+eyoFNKGETl+tg2Ruevh7tuVgMLD6jFhZwkm7hkS
eoG4h22rMYQ0EEJc/rjYuwT/xZySOhIIwbpg/TdDtoQi7j4Cru0b5BibZkcfRxQDhzqIvAPi7U113i7qPq/
jiC2fqnRoxl2lVuPuwCyEhDnWHatBT/
oIPEpJfvg7+zvnGk3PHvlpH3G6A85oEWFa8tbCkCt9ca8exwJCmbNCEbBnWL/
tl75HOGNfTweqNxoggQNbLEG8mmxDu9t5e/
LSDyeDycJ0CE2f8kFh+E7RGw6xznyUtS9c0CasOi0kAeXZ9fm1JRrYlAR+ADYECvRYmQEOfgpio/
2+vjTmBU0rmTSZkoNY5Ijw+SYljCzgjc9JX9Rt+EpRqnYm3BFBCg0yzW2bLyKFOZSYzbuHr0NXpJY50V04An
/Glj6Cv/vjPE8wUTkUecRBwYuyezNBPY7m7AU6sd1hTMcL3sDTJ2pzEJT56wKcV9zjnoQyBmrwatPjpU----
-END CERTIFICATE REQUEST-----"
                authenticate-client true
                cipher-list [
                    aes128-sha256
                    des-cbc3-sha
                    camellia128-sha
                ]
         }
     }
 }
```

6.3.2 Generating a self-signed certificate

Procedure

From the SR Linux CLI, you can create a self-signed certificate and key. By default, SHA256 with RSA encryption are used for the signature algorithm. Private keys are not encrypted using DES/3, and are 4096 bits in length.

Example:

The following example generates a private key, followed by the self signed certificate:

```
# tools system tls generate-self-signed email info@nokia.com country us organization nokia
/system/tls:
-----BEGIN PRIVATE KEY-----
MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQC3Q7PnCWjevlFn
--snip--
1ecXRjvpRvIEDUWYqVOioMfCaWxJoTJUX6HgjIY0Z9ktfWiceX1Ka4e3ZxIpEC4p
SvzKsoCTqpQcIk5pCsALhznOO6ArtPc4
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIE/TCCAuWgAwIBAgIJAKzAUREbPIV1MA0GCSqGSIb3DQEBCwUAMBQxEjAQBgNV
--snip-
BTTcAiQnIIkdJ0niqE+ZcOneSgxP3dKEovWQ+Bh3ES2QLsqbDvBYjz4eBoDigaAZ
KDnK207h3hiKLAaxMbAQeWGiu2ZKoKKdd4QE6OphOw6T
-----END CERTIFICATE-----
```

This **tools** command example is equivalent to the following **openssl** command:

```
# openssl req -x509 -newkey rsa:4096 -days 365
```

6.3.3 Generating a certificate signing request

Procedure

You can issue a CLI command that returns a private key and certificate signing request (CSR). This CSR can be passed to a certificate authority (the same one that the client/server uses to validate certificates on either side) for the certificate authority to sign the request; the CSR cannot be used as-is.

Example:

The following command returns the private key, followed by the CSR:

```
# tools system tls generate-csr email info@nokia.com country us organization nokia
/system/tls:
-----BEGIN PRIVATE KEY-----
MIIJRAIBADANBgkqhkiG9w0BAQEFAASCCS4wggkqAgEAAoICAQC3Q7PnCWjevlFn
--snip--
1ecXRjvpRvIEDUWYqVOioMfCaWxJoTJUX6HgjIY0Z9ktfWiceX1Ka4e3ZxIpEC4p
SvzKsoCTqpQcIk5pCsALhznOO6ArtPc4
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE REQUEST-----
MIIC5TCCAc0CAQAwgZ8xCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlh
vy3MjE7rJtmWTg0pTfiuU4BFoAzLhHRN1hl1mXuE2m6XJ8gvBPp2sN7SvieUCy/L
RVTs+/Fmcc4vMjx3t/0hAewIsd7DNe+kVQ==
-----END CERTIFICATE REQUEST-----
```

This **tools** command example is equivalent to the following **openssl** command:

```
# openssl req -newkey rsa:4096
```

7 Logging

The SR Linux device implements logging via the standard Linux syslog libraries. The SR Linux device uses rsyslog in the underlying Linux OS to filter logs and pass them on to remote servers or other specified destinations.

The main configuration file for rsyslog is `/etc/rsyslog.conf`. The SR Linux installs a minimal version of the `/etc/rsyslog.conf` file, and maintains an SR Linux-specific configuration file in the `/etc/rsyslog.d/` directory. Per-application logging configuration files are also kept in the `/etc/rsyslog.d/` directory; these files are named `/etc/rsyslog.d/`*`nn-app`*`.conf`.

You can modify the SR Linux logging configuration using the CLI, northbound API, or by editing the `.conf` files manually. The SR Linux device overwrites any parts of the configuration that are owned by SR Linux, and this may supersede any configuration done manually.

The SR Linux `.conf` files in `/etc/rsyslog.d` use standard rsyslog syntax for configuring filters and actions within rules.

The SR Linux supports configuration of Linux facilities and SR Linux subsystems as sources for log messages to filter. See the *SR Linux Log Events Guide* for properties and descriptions of the log messages that can be generated by SR Linux subsystems.

Basic logging configuration consists of specifying a source for input log messages, filtering the log messages, and specifying an output destination for the filtered log messages.

7.1 Input sources for log messages

The SR Linux supports using messages logged to Linux syslog facilities and messages generated by SR Linux subsystems as input sources for log messages.

Table 2: Linux syslog facilities describes the Linux syslog facilities that can be used as input sources for log messages.

Table 2: Linux syslog facilities

Facility	Description
auth	Security/authorization messages that do not contain secret information
authpriv	Security/authorization messages that may contain secret information
cron	Messages generated by cron
daemon	Messages generated by system daemons without their own facility
ftp	Messages generated by an FTP daemon
kern	Messages generated by the kernel
local0	Local use 0

Facility	Description
local1	Local use 1
local2	Local use 2
local3	Local use 3
local4	Local use 4
local5	Local use 5
local6	Local use 6
local7	Local use 7
lpr	Messages generated by the line printer subsystem
mail	Messages generated by a mail client or server
news	Messages generated by the network news subsystem
syslog	Messages generated internally by syslog
user	Messages generated by a user
uucp	Messages generated by the UUCP (UNIX-to-UNIX copy) subsystem

Table 3: SR Linux logging subsystem names lists the SR Linux subsystems that produce messages that can serve as input sources for log messages. By default, SR Linux subsystem messages are logged to Linux syslog facility local6.

Table 3: SR Linux logging subsystem names

Subsystem	Description
aaa	Messages generated by the aaa_mgr application (not including accounting messages)
accounting	Accounting messages generated by the aaa_mgr application
acl	Messages generated through an ACL log action
app	Messages generated by the aaa_mgr application
arpnd	Messages generated by the arp_nd_mgr application
bfd	Messages generated by the bfd_mgr application
bgp	Messages generated by the bgp_mgr application
chassis	Messages generated by the chassis_mgr application
fib	Messages generated by the fib_mgr application

Subsystem	Description
gnmi	Messages generated by the gnmi_server application
json	Messages generated by the json_rpc_server application
linux	Messages generated by the linux_mgr application
lldp	Messages generated by the lldp_mgr application
mgmt	Messages generated by the mgmt_svr application
mpls	Messages generated by the mpls_mgr application
netinst	Messages generated by the net_inst_mgr application
policy	Messages generated by the policy_mgr application
sdk	Messages generated by the sdk_mgr application
staticroute	Messages generated by the static_route_mgr application
xdp	Messages generated by the xdp_mgr application

7.2 Filters for log messages

You can configure filters to target specific messages or groups of log messages captured within the input message source.

A filter can specify the set of messages generated by a Linux facility at a specified priority; for example, messages generated by the kernel that have a priority of warning or higher, or mail facility messages that have a priority of critical.

Filtering can be done for messages generated by a specific SR Linux subsystem; for example, messages generated by the aaa_mgr application, or messages generated by the chassis_mgr application. SR Linux subsystem messages go to a specified Linux facility (by default, local6), and you can create filters for subsystem-specific messages from this facility.

Table 4: Logging priorities describes the logging priorities in order of severity.

Table 4: Logging priorities

Code	Priority name	Description
0	emergency	System is unusable
1	alert	Action must be taken immediately
2	critical	Critical conditions
3	error	Error conditions
4	warning	Warning conditions

Code	Priority name	Description
5	notice	Normal but significant conditions
6	informational	Informational messages
7	debug	Debug-level messages

7.3 Output destinations for log messages

You can set the action that the SR Linux takes for input messages that meet the criteria specified in a filter. This can include sending the messages to a destination such as a log file, the console, or a remote host.

For example, you can configure the SR Linux to send messages generated by the kernel that have priority warning to a file called `/var/log/srlinux/file/kernel-warning`.

Messages generated by an SR Linux subsystem, such as the bgp_mgr or gnmi_server application, can be sent to specified destinations.

Actions for messages matching a filter can include the following:

- Send the messages to a specified file in the `/var/log/srlinux/file/` directory.
- Send the messages to a buffer. A buffer is similar to a file, but uses memory as storage and is not persistent across system reboots. Messages sent to a buffer are stored in the `/var/log/srlinux/buffer/` directory.
- Send the messages to the console; that is, the Linux device `/dev/console`, which may be assigned to a serial device in hardware.
- Send the messages to one or more remote servers. You can specify a network-instance where rsyslogd is run and which serves as the source for the messages.

7.4 Defining filters

Procedure

Filters target specific messages or groups of log messages within the input message source. Filter criteria for log messages can include the following:

- Specific text within a message
- Prefix text at the beginning of a message
- Linux facility that generated the message
- Regular expression matching text within the message
- Syslog tag of the message

Example:

The following example shows a configuration that creates a filter that matches messages from the Linux facility kern that have a priority of warning or higher. See Table 4: Logging priorities for a list of logging priorities.

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         filter f1 {
                facility kern {
                    priority {
                        match-above warning [
                    }
                }
         }
     }
 }
```

Example:

The following example creates a filter that matches messages from the Linux facility local6 (where SR Linux subsystem messages are logged by default) that have a priority of informational or higher and contain the text `accounting`. This filter can be used to match messages from the SR Linux accounting subsystem.

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         filter f2 {
                contains accounting {
                facility local6 {
                    priority {
                        match-above informational [
                    }
                }
         }
     }
 }
```

7.5 Logging destination configuration

You can configure the SR Linux to send logging information to the following destinations:

- A log file
- Memory buffer storage
- The console (`/dev/console`)
- One or more remote servers

7.5.1 Specifying a log file destination

Procedure

The SR Linux can send log messages to a specified log file. By default, the log file resides in the `/var/log/srlinux/file` directory. You can specify the retention policy for the file, including the maximum size (default 10 Mb), as well as the number of files to keep in the rotation (default 4 files).

Example:

The following example uses messages from Linux facility cron as input, filters the messages for those that have critical priority, and sends the filtered messages to the file `/var/log/srlinux/file/cron-critical`:

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         file cron-critical {
                facility cron {
                    priority {
                        match-exact [
                            critical
                        ]
                    }
                }
         }
     }
 }
```

Example

The following example sends messages matching criteria specified in filter f1 to the file `/var/log/srlinux/file/f1-match`:

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         file f1-match {
                filter [
                    f1
                ]
         }
     }
 }
```

Example

The following example uses messages generated by the SR Linux AAA subsystem (that is, messages generated by the aaa_mgr application, but not including accounting messages) as input. The messages are filtered for those that have warning or informational priority, and the filtered messages are sent to the file `/var/log/srlinux/file/aaa-warn-info`.

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         file aaa-warn-info {
```

```
                subsystem aaa {
                    priority {
                        match-exact [
                            warning
                            informational
                        ]
                    }
                }
        }
    }
}
```

7.5.2 Specifying a buffer destination

Procedure

The SR Linux can send log messages to a buffer. A buffer is similar to a file, except that a buffer uses memory as storage and is not persistent across system reboots.

When the SR Linux device boots, it creates a non-swappable tmpfs virtual filesystem at `/var/log/srlinux/buffer`. This tmpfs filesystem has a fixed size of 512 Mb, which is reserved for buffer usage.

When a buffer is created through a commit transaction, the SR Linux verifies that there is enough buffer space available to contain all configured buffers based on their retention policies. If sufficient space is not available, the commit transaction fails.

Example:

The following example sends messages matching criteria specified in filter f1 to the buffer `/var/log/srlinux/buffer/f1-match`. A retention policy is specified so that when the buffer reaches 5,000,000 bytes, messages are written to a new buffer. After five buffers are filled, the oldest one is overwritten.

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
    logging {
        buffer f1-match {
               rotate 5
               size 5000000
               filter [
                   f1
               ]
        }
    }
}
```

7.5.3 Specifying the console as destination

Procedure

You can specify the console as a destination for log messages. The console refers to Linux device `/dev/console`, The console may be assigned to a serial device in hardware.

Example:

The following example uses messages generated by the SR Linux accounting subsystem as input, filters the messages for those that have informational priority or higher, and sends the filtered messages to `/dev/console`:

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         console {
                subsystem accounting {
                    priority {
                        match-above informational
                    }
                }
         }
     }
 }
```

7.5.4 Specifying a remote server destination

Procedure

The SR Linux can send log messages to one or more remote servers. You can specify the network-instance that the SR Linux uses to contact the remote servers. The rsyslogd process is run within the specified network-instance.

Example:

The following example uses messages generated by the SR Linux BGP subsystem (that is, messages generated by the bgp_mgr application) as input, filters the messages for those that have alert priority or higher, and sends the filtered messages to a remote server. The messages are sourced from the mgmt network-instance.

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         network-instance mgmt
         remote-server 1.1.2.2 {
                subsystem bgp {
                    priority {
                        match-above alert
                    }
                }
            }
        }
    }
 }
```

7.6 Specifying a Linux syslog facility for SR Linux subsystem messages

Procedure

All of the messages generated by SR Linux subsystems (see Table 3: SR Linux logging subsystem names) are logged to the same Linux syslog facility. This allows you to filter messages from all SR Linux subsystems by capturing logs from this facility.

By default, SR Linux subsystem messages are logged to the Linux syslog facility local6. You can optionally specify a different syslog facility. See Table 2: Linux syslog facilities for the syslog facilities.

Example:

The following example changes the Linux syslog facility where messages generated by SR Linux subsystems are logged from the default of local6 to local7:

```
--{ * candidate shared default }--[  ]--
# info system logging
 system {
     logging {
         subsystem-facility local7
     }
 }
```

8 Network-instances

On the SR Linux device, you can configure one or more virtual routing instances, known as "network-instances”. Each network-instance has its own interfaces, its own protocol instances, its own route table, and its own FIB.

When a packet arrives on a subinterface associated with a network-instance, it is forwarded according to the FIB of that network-instance. Transit packets are normally forwarded out another subinterface of the network-instance.

The SR Linux supports three types of network-instances: **default**, **ip-vrf**, and **mac-vrf**. Type **default** is the default network-instance and only one of this type is supported. Type **ip-vrf** is the regular network-instance; you can create multiple network-instances of this type.

Type **mac-vrf** functions as a broadcast domain and is associated with an **ip-vrf** network-instance via an Integrated Routing and Bridging (IRB) to support tunneling of Layer 2 traffic across an IP network. See mac-vrf network-instance.

Initially, the SR Linux has a default network-instance and no ip-vrf or mac-vrf network-instances.

A management network-instance, which isolates management network traffic from other network-instances configured on the device, is created by default, with the `mgmt0` port automatically added to it. See Configuring the management network-instance and the *SR Linux Interfaces Guide*.

8.1 Basic network-instance configuration

The following example creates network-instance "black” and associates two network subinterfaces and one loopback subinterface with it.

The configuration administratively enables the network-instance, specifies a description, and assigns it a router ID. The router ID is optional and is used as a default router identifier for protocols running within the network-instance.

The network-instance is configured to export routes and neighbors to the Linux routing table.

```
--{ candidate shared default }--[  ]--
# info network-instance black
 network-instance black {
        description "Sample network instance"
        type ip-vrf
        admin-state enable
        router-id 1.1.1.1
        interface ethernet-1/1.1 {
        }
        interface ethernet-1/2.1 {
        }
        interface lo0.1 {
        }
        protocols {
            linux {
                export-routes true
                export-neighbors true
            }
        ip-forwarding {
```

```
                receive-ipv4-check true
        }
}
```

In the preceding example, the **receive-ipv4-check** parameter is set to **true**; if an IPv4 packet is received on a subinterface of this network-instance, and the IPv4 operational status of the subinterface is down, then the packet is discarded. When the **receive-ipv4-check** parameter is set to **false**, IPv4 packets are received on all subinterfaces of this network-instance that are up, even if they do not have IPv4 addresses.

8.2 Path MTU discovery

Path MTU discovery is the technique for determining the MTU size on the network path between hosts. On the SR Linux, path MTU discovery is enabled by default for all network-instances, and can be manually enabled or disabled per network-instance.

If path MTU discovery is disabled, the system drops the MTU for the session to the number of bytes specified by the **min-path-mtu** parameter when an ICMP fragmentation-needed message is received; by default the **min-path-mtu** setting is 552 bytes.

```
--{ * candidate shared default }--[  ]--
# info network-instance black
   network-instance black  {
     mtu {
       path-mtu-discovery true
       min-path-mtu 552
      }
   }
```

8.3 Static routes

Within a network-instance, you can configure static routes. Each static route is associated with an IPv4 prefix or an IPv6 prefix, which represents the packet destinations matched by the static route. Each static route belongs to a specific network-instance. Different network-instances can have overlapping routes (static or otherwise) because each network-instance installs its own routes into its own set of route tables and FIBs.

Each static route must be associated with a statically configured next-hop group, which determines how matching packets are handled: either perform a blackhole discard action or a forwarding action. The next-hop group can specify a list of one or more next-hops (up to 128), each identified by an IPv4 or IPv6 address and a resolve flag. If the resolve flag is set to false, only a direct route can be used to resolve the IPv4 or IPv6 next-hop address; if the resolve flag is set to true, any route in the FIB can be used to resolve the IPv4 or IPv6 next-hop address.

Each static route has a specified metric and preference. The metric is the IGP cost to reach the destination. The preference specifies the relative degree this static route is preferred compared to other static and non-static routes available for the same IP prefix in the same network-instance.

A static route is installed in the FIB for the network-instance if the following conditions are met:

- The route has the lowest preference value among all routes (static and non-static) for the IP prefix.
- The route has the lowest metric value among all static routes for the IP prefix.

If BGP is running in a network-instance, all static routes of that same network-instance are automatically imported into the BGP local RIB, so that they can be redistributed as BGP routes, subject to BGP export policies.

You can use BFD to monitor reachability between the router and the configured next hops for a static route, making BFD sessions between the local router and the defined next hops a condition for an associated static route and next hops to be operationally active. See Configuring failure detection for static routes.

8.3.1 Configuring static routes

Procedure

To configure static routes, you specify route prefixes to point to next-hop groups, along with the metric and preference.

Example:

The following example configures IPv4 and IPv6 static route prefixes to point to next-hop groups and specifies a preference and metric for each one:

```
--{ * candidate shared default }--[ network-instance black ]--
# info static-routes
        static-routes {
            route 192.168.18.0/24 {
                admin-state enable
                metric 1
                preference 5
                next-hop-group static-ipv4-grp
            }
            route 2001:1::192:168:18:0/64 {
                admin-state enable
                metric 1
                preference 6
                next-hop-group static-ipv6-grp
            }
        }
```

Example

The following example configures the next-hop groups for the static routes:

```
--{ * candidate shared default }--[ network-instance black ]--
# info next-hop-groups
        next-hop-groups {
            group static-ipv4-grp {
                admin-state enable
                nexthop 1 {
                    ip-address 172.16.0.22
                }
                nexthop 2 {
                    ip-address 172.16.0.45
                    resolve true
                }
            }
            group static-ipv6-grp {
                admin-state enable
                blackhole {
                }
            }
```

```
        }
    }
```

In the preceding example, an IPv4 next-hop group is configured with two next-hops. The `resolve true` setting allows any route in the FIB to be used to resolve the IPv4 next-hop address, provided the resolution depth is not more than 2.

The IPv6 next-hop group is configured to perform a blackhole discard action for matching packets.

8.3.2 Configuring failure detection for static routes

Procedure

BFD can be used as a failure detection mechanism for monitoring the reachability of next hops for static routes. When BFD is enabled for a static route, it makes an active BFD session between the local router and the defined next hops required as a condition for a static route to be operationally active.

BFD is enabled for specific next-hop groups; as a result, BFD is enabled for any static route that refers to the next-hop group. If multiple next hops are defined within the next-hop group, a BFD session is established between the local address and each next hop in the next-hop group.

A static route is considered operationally up if at least one of the configured next-hop addresses can establish a BFD session. If the BFD session fails, the associated next hop is removed from the FIB as an active next hop.

Example:

The following example enables BFD for a static route next-hop:

```
--{ * candidate shared default }--[ network-instance black ]--
# info next-hop-groups
    next-hop-groups {
        group static-ipv4-grp {
            admin-state enable
            nexthop 1 {
                failure-detection {
                    enable-bfd {
                        local-address 1.2.34.5
                    }
                }
            }
        }
    }
```

A BFD session is established between the address configured with the **local-address** parameter and each next-hop address before that next-hop address is installed in the forwarding table.

All next-hop BFD sessions share the same timer settings, which are taken from the BFD configuration for the subinterface where the address in **local-address** parameter is configured. See BFD.

8.4 Aggregate routes

You can specify aggregate routes for a network-instance. Each aggregate route is associated with an IPv4 prefix or an IPv6 prefix, which represents the packet destinations matched by the aggregate route. As

with static routes, each aggregate route belongs to a specific network-instance, though different network-instances can have overlapping routes because each network-instance installs its own routes into its own set of route tables and FIBs.

An aggregate route can become active when it has one or more contributing routes. A route contributes to an aggregate route if all of the following conditions are met:

- The prefix length of the contributing route is greater than the prefix length of the aggregate route.
- The prefix bits of the contributing route match the prefix bits of the aggregate route up to the prefix length of the aggregate route.
- There is no other aggregate route that has a longer prefix length that meets the previous two conditions.
- The contributing route is actively used for forwarding and is not an aggregate route itself.

That is, a route can only contribute to a single aggregate route, and that aggregate route cannot recursively contribute to a less-specific aggregate route.

Aggregate routes have a fixed preference value of 130. If there is no route to the aggregate route prefix with a numerically lower preference value, then the aggregate route, when activated by a contributing route, is installed into the FIB with a blackhole next-hop. It is not possible to install an aggregate route into the route-table or as a BGP route without also installing it in the FIB.

The aggregate routes are commonly advertised by BGP or another routing protocol so that the individual contributing routes no longer need to be advertised.

This can speed up routing convergence and reduce RIB and FIB sizes throughout the network. If BGP is running in a network-instance, all active aggregate routes of that network-instance are automatically imported into the BGP local RIB so they can be redistributed as BGP routes, subject to BGP export policies.

8.4.1 Configuring aggregate routes

Procedure

The following example specifies an aggregate route. The aggregator address setting identifies the aggregating router. By default, this is the configured router ID of the BGP instance, or 0 if BGP is not enabled.

Example:

```
--{ * candidate shared default }--[ network-instance black ]--
# info aggregate-routes
        aggregate-routes {
            route 172.16.0.0/24 {
                aggregator {
                    address 1.1.1.1
                }
                summary-only true
                generate-icmp true
            }
        }
```

When the **summary-only** parameter is set to **true**, activation of an aggregate route automatically blocks the advertisement of all of its contributing routes by BGP.

The **generate-icmp true** setting causes the router to generate ICMP unreachable messages for the dropped packets.

8.5 Route preferences

A route can be learned by the router from different protocols, in which case, the costs are not comparable. When a route is learned from different protocols, the preference value is used to decide which route is installed in the forwarding table if several protocols calculate routes to the same destination. The route with the lowest preference value is selected.

Different protocols should not be configured with the same preference. If protocols are configured with the same preference, the tiebreaker is per the default preference table as defined in Table 5: Route preference defaults by route type. If multiple routes are learned with an identical preference using the same protocol, the lowest cost route is used.

Table 5: Route preference defaults by route type

Route Type	Preference	Configurable
Direct attached	0	No
Static routes	5	Yes
OSPF internal	10	Yes[1]
IS-IS level 1 internal	15	Yes
IS-IS level 2 internal	18	Yes
OSPF external	150	Yes
IS-IS level 1 external	160	Yes
IS-IS level 2 external	165	Yes
BGP	170	Yes

Note:

1. Preference for OSPF internal routes is configured with the **preference** command.

8.6 Displaying network-instance status

Procedure

Use the **show network-instance** command to display status information about network-instances configured on the device.

Example:

To display information about all configured network-instances, including the router ID, description, administrative and operational state:

```
--{ show }--
# show network-instance summary
```

```
+---------+---------+--------+-------+------------+--------------------------------+
|  Name   |  Type   | Admin  | Oper  | Router id  |   Description                  |
|         |         | state  | state |            |                                |
+=========+=========+========+=======+============+================================+
| default | default | enable | up    | 5.5.5.5    | Sample network instance        |
| mgmt    | ip-vrf  | enable | up    |            | Management network instance    |
| red     | ip-vrf  | enable | up    |55.55.55.55 | Network instance for bgp tests |
+---------+---------+--------+-------+------------+--------------------------------+
```

Example

To limit the display to a single network-instance:

```
--{ show }--
# show network-instance default summary
+---------+---------+--------+-------+---------+---------------------------+
|  Name   |  Type   | Admin  | Oper  | Router  |         Description       |
|         |         | state  | state |   id    |                           |
+=========+=========+========+=======+=========+===========================+
| default | default | enable | up    | 5.5.5.5 | "Sample network instance"  |
+---------+---------+--------+-------+---------+---------------------------+
```

Example

To display information about the interfaces attached to a network-instance:

```
--{ show }--
# show network-instance default interfaces
===============================================================
Net instance    : default
Interface       : ethernet-1/1.1
Oper state      : up
Ip mtu          : 1500
                  Prefix                      Origin        Status
  =============================================================
  192.35.1.0/31                             static
  2001:192:35:1::/127                       static       preferred
  fe80::201:5ff:feff:0/64                   link-layer   preferred
===============================================================
Net instance    : default
Interface       : lo0.1
Oper state      : up
                  Prefix                      Origin        Status
  =============================================================
  5.5.5.5/32                                static
  2001:5:5:5::5/128                         static       preferred
===============================================================
```

The command displays the operational state, IP MTU, and assigned IPv4/IPv6 prefix for each interface. If the operational state for an interface is down, the reason for the interface being down is shown.

8.7 mac-vrf network-instance

The network instance type **mac-vrf** is associated with a network-instance of type **default** or **ip-vrf** via an Integrated Routing and Bridging (IRB) interface.

The mac-vrf network-instance type functions as a broadcast domain. Each mac-vrf network-instance builds a bridge table composed of MAC addresses that can be learned via the data path on network-instance interfaces or via static configuration. You can configure the size of the bridge table for each mac-vrf network instance, as well as the aging for dynamically learned MAC addresses and other parameters related to the bridge table.

The mac-vrf network-instance type features a mac duplication mechanism that monitors MAC address moves across network-instance interfaces and across interfaces.

8.7.1 MAC selection

Each mac-vrf network-instance builds a bridge table to forward Layer 2 frames based on a MAC address lookup. The SR Linux selects the MAC addresses to be sent for installation to the line card (XDP), based on the following priority:

1. Local application MACs
2. Local static MACs
3. EVPN static (MACs coming from a MAC/IP route with the static bit set)
4. Local duplicate MACs
5. Learned or EVPN-learned MACs

8.7.2 MAC duplication detection and actions

MAC duplication is the mechanism used by SR Linux for loop prevention. MAC duplication monitors MAC addresses that move between subinterfaces. It consists of detection, actions, and process restarts.

8.7.2.1 MAC duplication detection

Detection of duplicate MAC addresses is necessary when extending broadcast domains to multiple leaf nodes. SR Linux supports a MAC duplication mechanism that monitors MAC address moves across network-instance interfaces.

A MAC address is considered a duplicate when its number of detected moves is greater than a configured threshold within a configured time frame where the moves are observed. Upon exceeding the threshold, the system holds on to the prior local destination of the MAC and executes an action.

8.7.2.2 MAC duplication actions

The action taken upon detecting one or more MAC addresses as duplicate on a subinterface can be configured for the mac-vrf network instance or for the subinterface. The following are the configurable actions:

- **oper-down** – When one or more duplicate MAC addresses are detected on the subinterface, the subinterface is brought operationally down.
- **blackhole** – upon detecting a duplicate mac on the subinterface, the mac will be blackholed.
- **stop learning** – Upon detecting a duplicate mac on the subinterface, the MAC address will not be relearned anymore on this or any subinterface. This is the default action for a mac-vrf network instance.

- **use-network-instance-action** – (Available for subinterfaces only) Use the action specified for the mac-vrf network instance. This is the default action for a subinterface.

8.7.2.3 MAC duplication process restarts

When at least one duplicate MAC address is detected, the duplicate MAC addresses are visible in the state datastore and can be displayed with the **info from state mac-duplication duplicate-entries** CLI command.

8.7.2.4 Configurable hold-down-time

The **info from state mac-duplication duplicate-entries** command also displays the hold-down-time for each duplicate MAC address. When the hold-down-time expires for all of the duplicate MAC addresses for the subinterface, the oper-down or stop-learning action is cleared, and the subinterface is brought operationally up or starts learning again.

The hold-down-time is configurable from between 2 and 60 minutes. You can optionally specify **indefinite** for the hold-down-time, which prevents the oper-down or stop-learning action from being cleared after a duplicate MAC address is detected; in this case, you can manually clear the oper-down or stop-learning action by changing the mac-duplication configuration or using the **tools network-instance bridge-table mac-duplication** command.

8.7.3 Bridge table configuration

The bridge table, its MAC address limit, and maximum number of entries can be configured on a per mac-vrf or per-subinterface basis.

When the size of the bridge table exceeds its maximum number of entries, the MAC addresses are removed in reverse order of the priority listed in MAC selection.

You can also configure aging for dynamically learned MAC addresses and other parameters related to the bridge table.

8.7.3.1 Deleting entries from the bridge table

Procedure

The SR Linux features commands to delete duplicate or learned MAC entries from the bridge table. For a mac-vrf or subinterface, you can delete all MAC entries, MAC entries with a blackhole destination, or a specific MAC entry.

Example:

The following example clears MAC entries in the bridge table for a mac-vrf network instance that have a blackhole destination:

```
--{ candidate shared default }--[  ]--
# tools network-instance mac-vrf-1 bridge-table mac-duplication delete-blackhole-macs
```

Example

The following example deletes a specified learned MAC address from the bridge table for a mac-vrf network instance:

```
--{ candidate shared default }--[  ]--
# tools network-instance mac-vrf-1 bridge-table mac-learning delete-mac 00:00:00:00:00:04
```

Example

The following example clears all duplicate MAC entries in the bridge table for a subinterface:

```
--{ candidate shared default }--[  ]--
# tools interface ethernet-1/1.1 bridge-table mac-duplication delete-all-macs
```

8.7.4 mac-vrf network instance configuration

The following example configures a mac-vrf network instance and settings for the bridge table. The bridge table is set to a maximum of 500 entries. Learned MAC addresses are aged out of the bridge table after 600 seconds.

MAC duplication detection is configured so that a MAC address is considered a duplicate when its number of detected moves across network instance interfaces is greater than 3 over a 5-minute interval. In this example, the MAC address is blackholed. After the hold-down-time of 3 minutes, the MAC address is flushed from the bridge table, and the monitoring process for the MAC address is restarted.

The example includes configuration for a static MAC address in the bridge table.

The mac-vrf network instance is associated with a bridged interface and an IRB interface.

```
--{ candidate shared default }--[  ]--
# info network-instance mac-vrf-1
 network-instance mac-vrf-1 {
        description "Sample mac-vrf network instance"
        type mac-vrf
        admin-state enable
        interface ethernet-1/1.1 {
        }
        interface irb1.1 {
        }
        bridge-table {
            mac-limit {
                mac-limit 500
            }
            mac-learning {
                admin-state enable
                aging {
                    admin-state enable
                    age-time 600
                }
            }
            mac-duplication {
                admin-state enable
                monitoring-window 5
                num-moves 3
                hold-down-time 3
                action blackhole
            static-mac {
                address [mac1
```

```
                }
            }
            }
        }
}
```

9 BFD

Bidirectional Forwarding Detection (BFD) is a lightweight mechanism used to monitor the liveliness of a remote neighbor. It is meant to be lightweight enough so that the ongoing sending and receiving mechanism can be implemented in the forwarding hardware. Because of this lightweight nature, BFD can send and receive messages at a much higher rate than other control plane hello mechanisms. This attribute allows connection failures to be detected faster than other hello mechanisms.

SR Linux supports BFD asynchronous mode, where BFD control packets are sent between two systems to activate and maintain BFD neighbor sessions between them.

BFD can be configured to monitor connectivity for the following:

- BGP peers. See Configuring BFD under the BGP protocol.
- Next-hops for static routes. See Configuring failure detection for static routes.
- OSPF adjacencies. See Configuring BFD under OSPF.
- IS-IS adjacencies. See Configuring BFD under IS-IS.

SR Linux supports one BFD session per port/connector, or up to 1152 sessions for an 8-slot chassis, depending on the hardware configuration.

On SR Linux systems that support link aggregation groups (LAGs), SR Linux supports micro-BFD, where BFD sessions are established for individual members of a LAG. If the BFD session for one of the links indicates a connection failure, the link is taken out of service from the perspective of the LAG. See Micro-BFD.

This section describes the minimal configuration necessary to set up BFD on SR Linux. To create a BFD session, you configure BFD on both systems (or BFD peers). When BFD has been enabled on the interfaces and at the global level for the appropriate routing protocols, a BFD session is created, BFD timers are negotiated, and the BFD peers begin to send BFD control packets to each other at the negotiated interval.

9.1 Configuring BFD for a subinterface

Procedure

You can enable BFD with an associated subinterface, and set values for intervals and criteria for declaring a session down.

Timer values are in microseconds. The detection interval for the BFD session is calculated by multiplying the value of the negotiated transmission interval by the value specified in this field.

Example:

The following example configures BFD for a subinterface.

```
--{ candidate shared default }--[  ]--
# info bfd
    bfd {
        subinterface ethernet-1/2.1 {
            admin-state enable
```

```
            desired-minimum-transmit-interval 250000
            required-minimum-receive 250000
            detection-multiplier 3
        }
    }
```

9.2 Configuring BFD under the BGP protocol

Procedure

BFD can be configured under the BGP protocol at the global, group, or neighbor level.

Before it is enabled, BFD must first be configured for a subinterface and timer values must be set. See Configuring BFD for a subinterface.

Example:

The following example configures BFD under the BGP protocol at the global level.

```
--{ candidate shared default }--[  ]--
# info network-instance default
    network-instance default {
        protocols {
            bgp {
                failure-detection {
                    enable-bfd true
                    }
                }
            }
        }
    }
```

Example

The following example configures BFD for the links between peers within an associated BGP peer group.

```
--{ candidate shared default }--[  ]--
# info network-instance default protocols bgp
    network-instance default {
        protocols {
            bgp {
                group test {
                    failure-detection {
                        enable-bfd true
                        }
                    }
                }
            }
    }
```

Example

The following example configures BFD for the link between BGP neighbors.

```
--{ candidate shared default }--[  ]--
# info network-instance default protocols bgp
    network-instance default {
```

```
        protocols {
            bgp {
                neighbor 192.35.1.3 {
                    failure-detection {
                        enable-bfd true
                        fast-failover true
                    }
                }
            }
        }
    }
```

9.3 Configuring BFD under OSPF

Procedure

For OSPF and OSPFv2, BFD can be enabled at the interface level to monitor the connectivity between the router and its attached network.

Example:

```
--{ candidate shared default }--[  ]--
# info network-instance default protocols ospf
    network-instance default {
        protocols {
            ospf {
                instance o1 {
                    area 1.1.1.1 {
                        interface ethernet-1/1.1 {
                            failure-detection {
                                enable-bfd true
                            }
                        }
                    }
                }
            }
        }
    }
```

9.4 Configuring BFD under IS-IS

Procedure

BFD can be configured at the interface level for IS-IS. You can optionally configure a BFD-enabled TLV to be included for IPv4 or IPv6 on the IS-IS interface.

Example:

```
--{ candidate shared default }--[  ]--
# info network-instance default protocols isis
    network-instance default {
        protocols {
            isis {
                instance i1 {
                    ipv4-unicast {
```

```
                    admin-state enable
                }
                interface ethernet-1/1.1 {
                    ipv4-unicast {
                        enable-bfd true
                        include-bfd-tlv true
                    }
                }
            }
        }
    }
```

9.5 Viewing the BFD state

Procedure

Use the **info from state** command to verify the BFD state for a network-instance.

Example:

```
# info from state bfd network-instance default peer 30
    bfd {
        network-instance default {
            peer 30 {
                oper-state up
                local-address 192.35.1.5
                remote-address 192.35.1.3
                remote-discriminator 25
                subscribed-protocols bgp_mgr
                session-state UP
                remote-session-state UP
                last-state-transition 2020-01-24T16:22:55.224Z
                failure-transitions 0
                local-diagnostic-code NO_DIAGNOSTIC
                remote-diagnostic-code NO_DIAGNOSTIC
                remote-minimum-receive-interval 1000000
                remote-control-plane-independent false
                active-transmit-interval 250000
                active-receive-interval 250000
                remote-multiplier 3
                async {
                    last-packet-transmitted 2020-01-24T16:23:19.385Z
                    last-packet-received 2020-01-24T16:23:18.906Z
                    transmitted-packets 32
                    received-packets 32
                    up-transitions 1
                }
            }
        }
    }
```

9.6 Micro-BFD

Micro-BFD refers to running BFD over the individual links in a link aggregation group (LAG) to monitor the bidirectional liveliness of the Ethernet links that make up the LAG.

A LAG member cannot be made operational within the LAG until the micro-BFD session is fully established. If a micro-BFD session fails, the corresponding Ethernet link is taken out of service from the perspective of the LAG.

Micro-BFD is supported on Ethernet LAG interfaces with an IP interface. Micro-BFD sessions are associated with each individual link. When enabled, the state of the individual links depends on the micro-BFD session state:

- Micro-BFD sessions must be established between both endpoints of a link before the link can be operationally up.
- If the micro-BFD session fails, the associated Ethernet link becomes operationally down from the perspective of the LAG.
- If LACP is not enabled for the LAG, if the Ethernet port is up, the system attempts to re-establish the micro-BFD session with the far end of the link.
- If LACP not enabled for the LAG, if the Ethernet port is up, the system attempts to re-establish the micro-BFD session with the far end of the link when LACP reaches distributing state.

If a link is not active for forwarding from the perspective of a LAG, ARP can still be performed across the link. For example, when a link is being brought up, and its micro-BFD session is not yet established, ARP can still be performed for the MAC address at the far end of the link, even though the link is not yet part of the LAG.

Micro-BFD packets bypass ingress and egress subinterface/interface ACLs, but received micro-BFD packets can be matched by CPM filters for filtering and logging.

Micro-BFD is supported on all SR Linux systems that also support LAGs: 7250 IXR; 7220 IXR-D1, D2, and D3; 7220 IXR-H2 and H3.

9.6.1 Configuring micro-BFD for a LAG interface

Procedure

The following example configures micro-BFD for a LAG interface. The example configures IP addresses to be used as the source address for IP packets and a remote address for the far end of the BFD session.

The example configures the minimum interval in microseconds between transmission of BFD control packets, as well as the minimum acceptable interval between received BFD control packets. The detection-multiplier setting specifies the number of packets that must be missed to declare the BFD session as down.

Example:

```
--{ * candidate shared default }--[  ]--
# info bfd micro-bfd-sessions
    micro-bfd-sessions {
        lag-interface lag1 {
            admin-state enable
            local-address 192.35.2.5
            remote-address 192.35.2.3
            desired-minimum-transmit-interval 250000
            required-minimum-receive 250000
            detection-multiplier 3
        }
    }
```

9.6.2 Viewing the micro-BFD state

Procedure

Use the **info from state** command to verify the micro-BFD state for members of a LAG interface.

Example:

```
# info from state micro-bfd-sessions lag-interface lag1 member-interface ethernet 2/1
    micro-bfd-sessions
        lag-interface lag1 {
            admin-state UP
            local-address 192.35.1.5
            remote-address 192.35.1.3
            desired-minimum-transmit-interval 250000
            required-minimum-receive 250000
            detection-multiplier 3
            member-interface ethernet 2/1 {
                session-state UP
                remote-session-state UP
                last-state-transition 2020-01-24T16:22:55.224Z
                last-failure-time 2020-01-24T16:22:55.224Z
                failure-transitions 0
                local-discriminator 25
                remote-discriminator 25
                local-diagnostic-code NO_DIAGNOSTIC
                remote-diagnostic-code NO_DIAGNOSTIC
                remote-minimum-receive-interval 1000000
                remote-control-plane-independent false
                active-transmit-interval 250000
                active-receive-interval 250000
                remote-multiplier 3
                async {
                   last-clear 2020-01-23T16:21:19.385Z
                   last-packet-transmitted 2020-01-24T16:23:19.385Z
                   last-packet-received 2020-01-24T16:23:18.906Z
                   transmitted-packets 32
                   received-errored-packets 3
                   received-packets 32
                   up-transitions 1
                }
            }
        }
    }
```

10 SR Linux applications

The SR Linux is a suite of modular, lightweight applications running like any others in a Linux environment. Each SR Linux application supports a different protocol or function, such as BGP, LLDP, AAA, and so on. These applications use gRPC and APIs to communicate with each other and external systems over TCP.

One SR Linux application, the application manager (app_mgr), is itself responsible for monitoring the health of the process IDs running each SR Linux applications, and restarting them if they fail. The application manager reads in application-specific YAML configuration and YANG models, and starts each application (or allows an application not to start if there no configuration exists for it). There is an instance of the app_mgr that handles applications running on the CPM, and an instance of the app_mgr on each IMM that handles applications running on the line card.

In addition to the Nokia-provided SR Linux applications, the SR Linux supports installation of user-defined applications, which are managed and configured in the same way as the default SR Linux applications. User-defined SR Linux applications can be loaded, reloaded, and unloaded from the system as necessary.

10.1 Installing an application

About this task

To install an application, copy the application files into the appropriate SR Linux directories, then reload the application manager and start the application.

The example in this topic installs an application called `fib_agent`. The application consists of files named `fib_agent.yml`, `fib_agent.sh`, `fib_agent.py`, and `fib_agent.yang`. The `fib_agent.yml` file is installed in the `/etc/opt/srlinux/appmgr/` directory. The `.yml` file for a user-defined application must reside in this directory in order for the app_mgr to read its YAML configuration.

The `.yml` file defines the locations of the other application files. The other application files can reside anywhere in the system other than in the `/opt/srlinux/` directory or any tempfs file system.

In this example, the `fib_agent.sh` and `fib_agent.py` files are installed in the directory `/user_agents/`, and the `fib_agent.yang` file is installed in the directory`/yang/`. The locations for these files are defined in the `fib_agent.yml` file.

Procedure

Step 1. Copy the application files into the SR Linux directories.

Example

```
--{ candidate }--[  ]--
# bash
# cp fib_agent.yml /etc/opt/srlinux/appmgr/.
# cp fib_agent.sh /user_agents/.
# cp fib_agent.py /user_agents/.
# cp fib_agent.yang /yang/.
# exit
```

Step 2. From the SR Linux CLI, reload the application manager.

Example

```
--{ candidate }--[ ]--
# tools system app-management application app_mgr reload
```

Step 3. Apply the changes to the configuration.

Example

```
--{ candidate }--[  ]--
# fib-agent
--{ candidate }--[ fib-agent ]--
# commit stay
All changes have been committed. Starting new transaction.
--{ candidate }--[ fib-agent ]--
```

Step 4. Verify that the application is running.

Example

```
# show system application fib_agent
  +-----------+-----+---------+----------------------+--------------------------+
  |  Name     | PID |  State  |        Version       |       Last Change        |
  +===========+=====+=========+======================+==========================+
  | fib_agent | 227 | running | v21.3.0-61-gd19567393 | 2021-01-13T20:16:45.697Z |
  +-----------+-----+---------+----------------------+--------------------------+
```

10.2 Starting an application

Procedure

To start an SR Linux application instance, use the **start** option in the **tools system app-management** command.

Example

To start an SR Linux application instance:

```
--{ running }--[  ]--
# tools system app-management application mpls_mgr start
/system/app-management/application[name=mpls_mgr]:
    Application 'mpls_mgr' was started
```

10.3 Restarting applications

SR Linux applications that are currently running can be restarted; that is, stopped then started again. SR Linux supports two types of application restarts: warm restart and cold restart. The difference between the two types involves how the application state is maintained in the IDB:

- Warm restart

  Warm restart allows forwarding to continue based on the previous state of the application. A warm restart causes the application to leave its state information in IDB during the restart, then recover it after

the restart. This restart type results in less disruption to surrounding applications that depend on the restarting application's state.

- Cold restart

  Cold restart results in a normal stop and start of the application, including purging its state in IDB.

Not all SR Linux applications support warm restart. For a list of applications that support warm restart, see the *SR Linux Release Notes*. You can also determine if an application supports warm restart by using an **info from state** command; for example:

```
# info from state system app-management application chassis_mgr supported-restart-types
    system {
        app-management {
            application chassis_mgr {
                supported-restart-types [
                    cold
                    warm
                ]
            }
        }
    }
```

Applications can be restarted automatically by app_mgr (for example, if an application fails and needs to be restarted) or you can restart an application manually from the CLI using a **tools** command. When app_mgr restarts an application, the application is warm-restarted if the application supports it; otherwise the application is cold-restarted. If you restart an application using a **tools** command, you can specify whether the application is cold or warm restarted. If you do not specify the restart type, the application is warm-restarted if the application supports it; otherwise the application is cold-restarted.

10.3.1 Restarting an application

Procedure

You can restart an application from the SR Linux CLI. The application can be cold-restarted or warm-restarted (if the application supports warm-restart). Using an **info from state** command, you can display the type of restart (cold or warm) that was used the last time the application was restarted.

Example: Restart an SR Linux application instance

```
--{ running }--[  ]--
# tools system app-management application chassis_mgr restart cold
/system/app-management/application[name=chassis_mgr]:
    Application 'chassis_mgr' was killed with signal 9

/system/app-management/application[name=chassis_mgr]:
    Application 'chassis_mgr' was restarted
```

Example: Display the type of restart used the last time the application was restarted

```
--{ running }--[  ]--
# info from state system app-management application chassis_mgr last-start-type
    system {
        app-management {
            application chassis_mgr {
                last-start-type cold
            }
        }
```

```
    }
```

10.4 Terminating an application

Procedure

You can use the **stop**, **quit**, or **kill** options in the **tools system app-management** command to terminate an SR Linux application.

- **stop** gracefully terminates the application, allowing it to clean up before exiting.
- **quit** terminates the application and generates a core dump. The core dump files are saved in the `/var/log/srlinux/cores/` directory.
- **kill** terminates the application immediately, without allowing it to clean up before exiting.

Example:

To terminate an application gracefully:

```
--{ candidate }--[  ]--
# tools system app-management application mpls_mgr stop
/system/app-management/application[name=mpls_mgr]:
    Application 'mpls_mgr' was killed with signal 15
```

Example

To terminate an application and generate a core dump:

```
--{ candidate }--[  ]--
# tools system app-management application mpls_mgr quit
/system/app-management/application[name=mpls_mgr]:
    Application 'mpls_mgr' was killed with signal 3
```

Example

To terminate an application immediately:

```
--{ candidate }--[  ]--
# tools system app-management application mpls_mgr kill
/system/app-management/application[name=mpls_mgr]:
    Application 'mpls_mgr' was killed with signal 9
```

10.5 Reloading application configuration

Procedure

Reloading the application manager causes it to process any changes in the installed applications' YAML configuration and restart the applications. Applications that are no longer present in the system are stopped, and their YANG modules are removed from the management server. Applications removed from the system have their nodes or augmentations removed from the system schema.

Example:

To reload the application configuration, reload the app_mgr application:

```
--{ * running }--[  ]--
# tools system app-management application app_mgr reload
```

10.6 Clearing application statistics

Procedure

You can display statistics collected for an application with the **info from state** command. To reset the statistics counters for the application, use the **statistics clear** option in the **tools system app-management** command.

Example:

To reset the statistics counters for an application:

```
--{ running }--[  ]--
# tools system app-management application mpls_mgr statistics clear
```

10.7 Restricted operations for applications

An application may have one or more operations that are restricted by default. For example, the linux_mgr application has stop, quit, and kill as restricted operations, meaning that these options are not available when entering the **tools system app-management** command for the linux_mgr application.

Table 6: Restricted operations for SR Linux applications lists the restricted operations for each SR Linux application.

Table 6: Restricted operations for SR Linux applications

Application	Restricted operations
aaa_mgr	reload
acl_mgr	reload
app_mgr	start, stop, restart, quit, kill
arp_nd_mgr	reload
bfd_mgr	reload
bgp_mgr	reload
chassis_mgr	stop, quit, kill, reload
device_mgr	reload

Application	Restricted operations
dhcp_client_mgr	stop, reload
fib_mgr	reload
gnmi_server	reload
idb_server	start, stop, restart, quit, kill, reload
json_rpc_config	reload
linux_mgr	stop, quit, kill
lldp_mgr	reload
log_mgr	reload
mgmt_server	start, stop, quit, kill, reload
mpls_mgr	reload
net_inst_mgr	start, stop, quit, kill, reload
oam_mgr	reload
plcy_mgr	reload
qos_mgr	reload
sdk_mgr	reload
static_route_mgr	reload
supportd	reload
xdp_cpm	stop, quit, kill, reload
xdp_lc	reload

Restricted options are specified in the `restricted-operations` setting in the YAML file for the application.

10.8 Configuring an application

About this task

To configure an SR Linux application, edit settings in the application YAML file, then reload the application manager to activate the changes.

The example in this section shows how to configure an application to specify the action the SR Linux device takes when the application fails. If an SR Linux application fails a specified number of times over a specified time period, the SR Linux device can reboot the system or attempt to restart the application after waiting a specified number of seconds.

For example, if the aaa_mgr application crashes 5 times within a 500-second window, the SR Linux device can be configured to wait 100 seconds, then restart the aaa_mgr application.

The following actions can be taken if an SR Linux application fails:

- Reboot the system
- Wait a specified number of seconds, then attempt to restart the failed application
- Move the failed application to error state without rebooting the system or attempting to restart the application

If you stop or restart an application using the **tools system app-management** command in the SR Linux CLI, it is not considered an application failure; the failure action for the application, if one is configured, does not occur. However, if the failed application waits a specified period of time (or forever) to be restarted, or has been moved into error state, you can restart the application manually with the **tools system app-management application restart** CLI command.

To configure the failure action for an application:

Procedure

Step 1. Check the status of the SR Linux applications:

Example

```
--{ running }--[  ]--
# show system application
+--------------+-----+--------------------+----------------------+--------------------------+
|    Name      | PID |       State        |        Version       |       Last Change        |
+==============+=====+====================+======================+==========================+
| aaa_mgr      | 242 | error              | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.967Z |
| acl_mgr      | 193 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.967Z |
| app_mgr      | 118 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:11.161Z |
| arp_nd_mgr   | 104 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.967Z |
| bfd_mgr      |     | waiting-for-config |                       |                          |
| bgp_mgr      | 109 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.156Z |
| chassis_mgr  | 115 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.967Z |
| dev_mgr      | 150 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.001Z |
| fib_mgr      | 168 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| gnmi_server  | 157 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.296Z |
| idb_server   | 171 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.228Z |
| igmp_mgr     |     | waiting-for-config |                       |                          |
| isis_mgr     |     | waiting-for-config |                       |                          |
| json_rpc     | 166 | running            |                       | 2022-01-21T20:15:13.234Z |
| label_mgr    | 139 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.186Z |
| lag_mgr      | 103 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| ldp_mgr      |     | waiting-for-config |                       |                          |
| linux_mgr    | 116 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| lldp_mgr     | 149 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.206Z |
| log_mgr      | 127 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| mgmt_server  | 149 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| mirror_mgr   |     | waiting-for-config |                       |                          |
| mpls_mgr     |     | waiting-for-config |                       |                          |
| net_inst_mgr | 160 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| oam_mgr      | 160 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.231Z |
| ospf_mgr     |     | waiting-for-config |                       |                          |
| p4rt_server  | 641 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T21:56:47.935Z |
| plcy_mgr     | 177 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.242Z |
| qos_mgr      | 186 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:13.322Z |
| sshd-mgmt    | 192 | running            |                       | 2022-01-21T20:15:19.710Z |
| xdp_cpm      | 197 | running            | v22.3.0-34-ge0ee326f8 | 2022-01-21T20:15:10.968Z |
| xdp_lc_1     | 108 | running            |                       | 2022-01-21T20:15:10.968Z |
```

```
+--------------+-----+--------------------+----------------------+-------------------------+
```

Step 2. Use the **info from state** command to check the current failure action settings for the application to configure. These settings are highlighted in the following example:

Example

```
--{ running }--[  ]--
# info from state system app-management application aaa_mgr
    system {
        app-management {
            application aaa_mgr {
                pid 242
                state error
                last-change 2022-01-21T20:15:10.967Z
                author Nokia
                failure-threshold 3
                failure-window 300
                failure-action reboot
                path /opt/srlinux/bin
                launch-command ./sr_aaa_mgr
                search-command ./sr_aaa_mgr
                version v22.3.0-34-ge0ee326f8
                restricted-operations [
                    reload
                ]
                statistics {
                    restart-count 0
                }
                yang {
                    modules [
                        srl_nokia-aaa
                        srl_nokia-aaa-types
                    ]
                    source-directories [
                        /opt/srlinux/models/ietf
                        /opt/srlinux/models/srl_nokia/models/common
                        /opt/srlinux/models/srl_nokia/models/network-instance
                        /opt/srlinux/models/srl_nokia/models/system
                    ]
                }
            }
        }
    }
```

The following failure action settings can be configured for an application:

- `failure-threshold`: number of times that the application must fail during the *failure-window* period before the *failure-action* is taken; the default is three times.
- `failure-window`: number of seconds over which the application must fail the *failure-threshold* number of times before the *failure-action* is taken; the default is 300 seconds.
- `failure-action`: action to take if the application fails *failure-threshold* times over *failure-window* seconds. This can be one of the following:
  - `reboot`: reboot the system; this is the default *failure-action*.
  - `wait=`*`seconds`*: wait this number of seconds, then attempt to restart the application.
  - `wait=forever`: move the application to error state and do not reboot the system or attempt to restart the application.

Step 3. Edit the YAML configuration for the application.

The YAML configuration files for SR Linux applications are located in the directory `/opt/srlinux/appmgr` on the SR Linux device. They are named `sr_application_name_config.yml`; for example, `sr_aaa_mgr_config.yml`.

Step 4. In the `.yml` file, add or change the settings for the `failure-threshold`, `failure-window`, and `failure-action` parameters.

Example

In the following example, the failure action settings in the `sr_aaa_mgr_config.yml` file are configured so that if the aaa_mgr application fails 5 times over a 500-second period, the SR Linux device waits 100 seconds, then attempts to restart the aaa_mgr application:

```
aaa_mgr_setup:
   path: /opt/srlinux/bin
   launch-command: ./aaamgr_set_env.sh
   run-as-user: root
   never-show: Yes
   never-restart: Yes
   start-order: 1
aaa_mgr:
   path: /opt/srlinux/bin
   launch-command:  ./sr_aaa_mgr
   search-command: ./sr_aaa_mgr
   run-as-user: root
   restricted-operations: ['reload']
   failure-threshold: 5
   failure-window : 500
   failure-action: "wait=100"
   yang-modules:
       names:
           - "srl_nokia-aaa"
           - "srl_nokia-aaa-types"
       source-directories:
           - "/opt/srlinux/models/ietf"
           - "/opt/srlinux/models/srl_nokia/models/common"
           - "/opt/srlinux/models/srl_nokia/models/system"
           - "/opt/srlinux/models/srl_nokia/models/network-instance"
```

Step 5. Save and close the `.yml` configuration file.

Step 6. In the SR Linux CLI, reload the application manager:

Example

```
--{ running }--[  ]--
# tools system app-management application app_mgr reload
```

This command reloads any application whose `.yml` configuration file has changed. It does not affect any service.

Step 7. Use the **info from state** command to verify that the changes to the failure action settings are now in effect.

Example

```
--{ running }--[  ]--
# info from state system app-management application aaa_mgr
    system {
        app-management {
```

```
            application aaa_mgr {
                pid 242
                state running
                last-change 2022-01-21T20:15:10.967Z
                author Nokia
                failure-threshold 5
                failure-window 500
                failure-action wait=100
                path /opt/srlinux/bin
                launch-command ./sr_aaa_mgr
                search-command ./sr_aaa_mgr
                version v22.3.0-34-ge0ee326f8
                restricted-operations [
                    reload
                ]
                statistics {
                    restart-count 0
                }
                yang {
                    modules [
                        srl_nokia-aaa
                        srl_nokia-aaa-types
                    ]
                    source-directories [
                        /opt/srlinux/models/ietf
                        /opt/srlinux/models/srl_nokia/models/common
                        /opt/srlinux/models/srl_nokia/models/network-instance
                        /opt/srlinux/models/srl_nokia/models/system
                    ]
                }
            }
        }
    }
```

10.9 Removing an application from the system

About this task

To remove an application from the system, remove the application files from the SR Linux directories where they reside, then reload the application manager.

When an application is removed from the system, SR Linux stops sending updates for the paths that the application would populate. If the application is subsequently reloaded, SR Linux resumes sending updates.

For active CLI sessions, the schema is updated to remove the application. If an active CLI session exists in a context that is no longer present because of the application being removed, the CLI context is changed to the next-highest valid context.

User-defined applications can be removed from the system, but removing Nokia-provided SR Linux applications is not supported.

In the following example, the `fib_agent` application, consisting of files named `fib_agent.yml`, `fib_agent.sh`, `fib_agent.py`, and `fib_agent.yang`, is removed from the system.

Procedure

Step 1. Remove the application files from the SR Linux directories.

Example

```
--{ candidate }--[  ]--
# bash
# rm /etc/opt/srlinux/appmgr/fib_agent.yml
# rm /user_agents/fib_agent.sh
# rm /user_agents/fib_agent.py
# rm /yang/fib_agent.yang
# exit
```

Step 2. From the SR Linux CLI, reload the application manager.

Example

```
--{ candidate }--[ ]--
# tools system app-management application app_mgr reload
```

The application manager stops any application that is no longer present, and removes the application's YANG module from the management server.

10.10 Partioning and isolating application resources

The SR Linux protects system processes through the use of control groups (cgroups), which impose resource consumption limits on resource-intensive customer applications.

10.10.1 Cgroup profiles

Cgroup profiles define how usage limits are applied. On the SR Linux, cgroup profiles are supported for CPU and memory and are defined in cgroup_profile.json configuration files.

SR Linux provides a default cgroup profile; customers can configure additional cgroup profiles.

10.10.1.1 Default cgroup profile

The SR Linux-provided default cgroup profile is located in the `/opt/srlinux/appmgr/cgroup_profile.json` directory.

Note: Editing the default cgroup profile is not recommended.

If the default cgroup profile fails to parse or be read by the app_mgr, the SR Linux does not start.

The default cgroup_profile.json file definition is shown below:

```
{
 "profiles": [
  {
   "name": "workload.primary",
   "path": "workload.slice/primary",
   "controller": {
    "memory": {
     "max": 0.8,
     "swap_max": 0,
```

```
    "low": 0
   },
   "cpu": {
    "weight": "10000",
    "period": "100000",
    "quota": "0"
   },
   "cpuset": {
    "cpus": "",
    "mems": ""
   }
  }
 },
 {
  "name": "workload.datapath",
  "path": "workload.slice/datapath",
  "controller": {
   "memory": {
    "max": 0.8,
    "swap_max": 0,
    "low": 0
   },
   "cpu": {
    "weight": "10000",
    "period": "100000",
    "quota": "0"
   },
   "cpuset": {
    "cpus": "all",
    "mems": ""
   }
  }
 },
 {
  "name": "workload.secondary",
  "path": "workload.slice/secondary",
  "controller": {
   "memory": {
    "max": 0.5,
    "swap_max": 0,
    "low": 0
   },
   "cpu": {
    "weight": "10000",
    "period": "100000",
    "quota": "0"
   },
   "cpuset": {
    "cpus": "",
    "mems": ""
   }
  }
 },
 {
  "name": "user.default",
  "path": "user.slice/default",
  "controller": {
   "memory": {
    "max": 0.25,
    "swap_max": 0,
    "low": 0
   },
   "cpu": {
    "weight": "1000",
```

```
      "period": "100000",
      "quota": "0"
     },
     "cpuset": {
      "cpus": "",
      "mems": ""
     }
    }
   }
  ]
}
```

Table 7: Default cgroup profile parameters describes the default cgroup profile parameters.

Table 7: Default cgroup profile parameters

Parameter	Description
name	The cgroup profile name. Type: string
path	The cgroup directory path relative to a unified root path. A typical unified root path is /sys/fs/cgroup or /mnt/cgroup/unified Type: string
controller	The memory controller configuration. **max** This number denotes the percentage of total memory. The actual memory value is calculated as (max. × total_memory) and is set in the memory.max interface file of the cgroup. If the value is 0, this configuration is ignored. The range is from 0 to 1, the default is 0.8. **low** This number denotes the percentage of total memory. The actual memory value is calculated as (max. × total_memory) and is set in the memory.low interface file of the cgroup. The range is from 0 to 1, the default is 0.8.
cpu	The CPU controller configuration. **weight** This value is set in the cpu.weight interface file of the cgroup. The range is from 1 to 10 000, the default is 100. **period** This value specifies a period of time, in microseconds, for how regularly a cgroup's access to CPU resources should be reallocated.This value is set in the cpu.max interface file of the cgroup. The range is from 1000 to 1 000 000, the default is 100 000. **quota** This value specifies the total length of time, in microseconds, for which all tasks in a cgroup can run during one period (as defined in the **period** parameter). If quota is set to 0, it translates to "max" in the cpu.max interface file. The range is from 1000 to 1 000 000, the default is max.
cpuset	CPU usage information for the cgroup. **cpus** This value indicates the CPUs used by the cgroup. This can be " ", meaning use all CPUs except for the isolated CPUs; this is the default.

Parameter	Description
	The value `all` means include the isolated CPUs for cgroup usage. The value `x, y-z`, where `x`, `y`, and `z` are CPU numbers, means use a specific CPU or a range of CPUs. **mems** This value is used for scheduling multiple NUMA (non-uniform memory access) aware applications in the cgroup.

10.10.1.2 Customer-defined cgroup profile

Customers can configure cgroup profiles in the `/etc/opt/srlinux/appmgr/cgroup_profile.json` directory. The app_mgr creates this directory at boot up if it does not exist.

If a customer-defined cgroup profile fails to load, the system continues to function and applications that are loaded into the customer cgroup are loaded using Nokia defaults, listed below.

- Nokia-written applications run in the workload.slice/primary cgroup along with any processes that are started by linuxadmin, including sr_cli.
- Non-Nokia-written applications run in the workload.slice/secondary cgroup. If a customer builds an application and launches it using the app_mgr without specifying a cgroup, the application runs in this cgroup
- All interactive user applications run in the user.slice/default cgroup, including sr_cli when not started by linuxadmin.

The admin user is treated as any other user in the system. Its processes fall into the user.slice/default cgroup.

10.10.2 Configuring a cgroup

Customers can configure up to three cgroups in the `/etc/opt/srlinux/appmgr/cgroup_profile.json` directory. Customer applications are assigned to these groups. Any more than three configured cgroups are ignored. The depth of cgroups is limited to two levels where, for example, workload is one level, and primary/secondary are two levels. Any levels beyond this are also ignored.

If a cgroup with the same name is used in multiple customer-defined profiles, the system ignores it and uses the cgroup defined in the default profile.

10.10.2.1 Cgroup configuration example

About this task

The following example shows the configuration of two customer-defined cgroups: one for a lightweight database that needs priority access to resources, and one for storing the users of the database.

The steps and the outputs are described below.

The configuration above created two cgroup profiles: one for the database slice and one for the frontend slice. The profile for the database slice is configured to limit the database to 50% of system memory. The profile for the frontend slice is configured to limit the web front end to 20% of system memory.

In addition, both cgroup profiles are configured to limit CPU resources for their respective cgroup. The database server CPU is weighted at 10000 (the maximum CPU weight) and the frontend server CPU is weighted at 5000 (half the CPU weight of the database). The weights are added together and each group is allocated its ratio of CPU as a proportion of the sum. The periods are kept the same, and no guaranteed CPU is granted.

Procedure

Step 1. Install the application, including the YAML binary file and optional YANG module.

In this example, YAML defines two applications: dtw-database and dtw-frontend. These applications are placed into their own cgroups: distributetheweb.slice/database and distributetheweb.slice/frontend.

The app-mgr creates the cgroups.

The output below shows the installation of the database and a web front end.

```
dtw-database:
   path: /opt/distributetheweb/bin/
   launch-command: ./run_db
   search-command: ./run_db
   failure-threshold: 100
   failure-action: "wait=60"
   cgroup: distributetheweb.slice/database
   oom-score-adj: -500
   yang-modules:
       names:
           - "database"
       source-directories:
           - "/opt/distributetheweb/yang/"
```

```
dtw-frontend:
   path: /opt/distributetheweb/bin/
   launch-command: ./run_frontend
   search-command: ./run_frontend
   failure-threshold: 100
   failure-action: "wait=60"
   cgroup: distributetheweb.slice/frontend
   oom-score-adj: 200
   yang-modules:
       names:
           - "frontend"
       source-directories:
           - "/opt/distributetheweb/yang/"
```

Step 2. Configure the cgroup profiles in the `/etc/opt/srlinux/appmgr/cgroup_profile.json` directory.

The output below shows the configuration.

```
{
 "profiles": [
  {
   "name": "distributetheweb.database",
   "path": "distributetheweb.slice/database",
   "controller": {
    "memory": {
     "max": 0.5,
     "swap_max": 0,
     "low": 0
    },
```

```
    "cpu": {
     "weight": "10000",
     "period": "100000",
     "quota": "0"
    }
   }
  },
  {
   "name": "distributetheweb.frontend",
   "path": "distributetheweb.slice/frontend",
   "controller": {
    "memory": {
     "max": 0.2,
     "swap_max": 0,
     "low": 0
    },
    "cpu": {
     "weight": "5000",
     "period": "100000",
     "quota": "0"
    }
   }
  }
 ]
}
```

10.10.3 Kernel low-memory killer

The kernel low-memory killer driver monitors the memory state of a running system. It reacts to high memory pressure by killing the least essential processes to keep the system performing at acceptable levels.

When the system is low in memory and cannot find free memory space, the out_of_memory function is called. The out_of_memory function makes memory available by killing one or more processes.

When an out-of-memory (OOM) failure occurs, the out_of_memory function is called and it obtains a score from the select_bad_process function. The process with the highest score is the one that is killed. Some of the criteria used to identify a bad process include the following:

- The kernel needs a minimum amount of memory for itself.
- Try to reclaim a large amount of memory.
- Do not kill a process that is using a small amount of memory.
- Try to kill the minimum number of processes.
- Algorithms that elevate the sacrifice priority on processes the user wants to kill.

In addition to this list, the OOM killer checks the out-of-memory (OOM) score. The OOM killer sets the OOM score for each process and then multiplies that value by memory usage. The processes with higher values have a high probability of being terminated by the OOM killer.

10.10.3.1 SR Linux process kill strategy

The kernel calculates the oom_score using the formula 10 × percentage of memory used by the process. The maximum score is 10 × 100% = 1000. The oom_score of a process can be found in the `/proc` directory (`/proc/$pid/oom_score`). An oom_score of 1000 means the process is using all the memory,

an oom_score of 500 means it is using half the memory, and an oom_score of 0 means it is using no memory.

The OOM killer checks the `oom_score_adj` file in the `/proc/$pid/oom_score_adj` directory to adjust its final calculated score. The default value is 0.

The `oom_score_adj` value can range from -1000 to 1000. A score of -1000 results in a process using 100% of the memory and in not being terminated by the OOM killer. However, a score of 1000 causes the Linux kernel to terminate the process even when it uses minimal memory. A score of -100 results in a process using 10% of the memory before it is considered for termination, as its score remains 0 until its unadjusted score reaches 100.

The `oom_score_adj` value for each process is defined in its corresponding YAML definition file. The system groups the processes based on their score, which the SR Linux OOM killer uses as the hierarchy for terminating a rogue process, as follows:

- group1 = -998, for processes that should never be killed, such as app_mgr, idb_server, and all processes on the IMM.
- group2 = -200, for processes that ideally should not be killed because doing so has a substantial impact on the system, such as mgmt_server, chassis_mgr, and net_inst_mgr. A score of -200 means the process gets to use 20% of the memory before their memory use starts being counted.
- group3 = 0, for process that should be killed if they are using too much memory such as BGP, ISIS, and OSPF.
- group4 = 500, for processes that should be preferentially killed, such as cli and oam_mgr

Table 8: OOM adjust score per process lists the OOM adjust score for each process.

Table 8: OOM adjust score per process

Process name	OOM adjust score
aaa_mgr	0
acl_mgr	0
app_mgr	-998
sarp_nd_mgr	-200
bfd_mgr	-200
bgp_mgr	0
chassis_mgr	-200
dev_mgr	-200
dhcp_client_mgr	0
dhcp_relay_mgr	0
eth_switch_mgr	-200
evpn_mgr	0

Process name	OOM adjust score
fib_mgr	0
gnmi_server	500
idb_server	-998
isis_mgr	0
json_rpc	500
l2_mac_learn_mgr	0
l2_mac_mgr	0
l2_static_mac_mgr	0
lag_mgr	0
linux_mgr	0
lldp_mgr	0
log_mgr	0
mcid_mgr	0
mgmt_server	-200
mpls_mgr	0
net_inst_mgr	-200
oam_mgr	500
ospf_mgr	0
plcy_mgr	0
qos_mgr	0
sdk_mgr	500
sflow_sample_mgr	500
sshd-mgmt	0
static_route_mgr	0
supportd	0
timesrv	0
vrrp_mgr	0

Process name	OOM adjust score
vxlan_mgr	0
xdp_cpm	-200

10.10.4 Application manager extensions for cgroups

The cgroup feature provides two additional parameters in the app_mgr YAML file, the **cgroup** parameter and the **oom-score-adj** parameter.

The app_mgr uses the **cgroup** parameter to launch an application within a specific cgroup. A valid value for the **cgroup** parameter is the path of a cgroup as specified in the cgroup profile (equivalent to `/profiles/name[]/path`), from the cgroupv2 root. If this cgroup does not exist, the app_mgr launches the user application from the default cgroup profile path `workload.slice/secondary`.

The app_mgr uses the **oom-score-adj** parameter to set the out-of-memory adjust score for a process. This score is fed into the SR Linux OOM killer. Valid **oom-score-adj** scores are any value in the range of -1000 to 1000. A process with a score of -1000 is least likely to be killed while a process with a score of 1000 is most likely to be killed. At -1000, a process can use 100% of memory and still avoid being terminated by the OOM killer; however, SR Linux kills the process more frequently.

The **cgroup** parameter and the **oom-score-adj** parameter are shown in the output below.

```
bgp_mgr:
   path: @SRLINUX_BINARY_INSTALL_PREFIX@
   launch-command: @YAML_LAUNCH_ENVIRONMENT@ ./sr_bgp_mgr
   search-command: ./sr_bgp_mgr
   oom-score-adj: 0
   wait-for-config: Yes
   author: 'Nokia'
   restricted-operations: ['reload']
   cgroup: workload.slice/primary
   yang-modules:
       names:
           - "srl_nokia-bgp"
           - "srl_nokia-bgp-vpn"
           - "srl_nokia-rib-bgp"
           - "srl_nokia-system-network-instance-bgp-vpn"
       tools:
           - "srl_nokia-tools-bgp"
       source-directories:
           - "@SRLINUX_FILE_INSTALL_PREFIX@/models/ietf"
           - "@SRLINUX_FILE_INSTALL_PREFIX@/models/srl_nokia/models/common"
           - "@SRLINUX_FILE_INSTALL_PREFIX@/models/srl_nokia/models/interfaces"
           - "@SRLINUX_FILE_INSTALL_PREFIX@/models/srl_nokia/models/network-
instance"
           - "@SRLINUX_FILE_INSTALL_PREFIX@/models/srl_nokia/models/routing-policy"
           - "@SRLINUX_FILE_INSTALL_PREFIX@/models/srl_nokia/models/system"
```

10.10.5 Debugging cgroups

Cgroup debugging capability is available through:

- SR Linux CLI commands

- Linux-provided CLI commands

10.10.5.1 SR Linux cgroup debugging commands

The SR Linux provides CLI commands to perform the following:

- check the usage of existing cgroups
- show information about the OOM adjust score of applications managed by the app_mgr
- show information about the cgroups that are associated with the applications that are managed by the app_mgr
- list all of the applications associated with a specified cgroup

10.10.5.1.1 Checking existing cgroup usage

The output below is an example of checking existing cgroup usage.

```
--{ running }--[  ]--
# info from state platform control A cgroup *
    platform {
        control A {
            cgroup /mnt/cgroup/unified/user.slice/default {
                memory-statistics {
                    current 0
                    current-swap 0
                    anon 0
                    kernel-stack 0
                    slab 0
                    sock 0
                    anon-thp 0
                    file 0
                    file-writeback 0
                    file-dirty 0
                }
                cpuacct-statistics {
                    user 0
                    system 0
                }
            }
            cgroup /mnt/cgroup/unified/workload.slice/primary {
                memory-statistics {
                    current 1621110784
                    current-swap 0
                    anon 1327427584
                    kernel-stack 2396160
                    slab 12939264
                    sock 151552
                    anon-thp 855638016
                    file 250916864
                    file-writeback 0
                    file-dirty 0
                }
                cpuacct-statistics {
                    user 22082989551
                    system 11252275018
                }
            }
            cgroup /mnt/cgroup/unified/workload.slice/secondary {
```

```
                memory-statistics {
                    current 7827456
                    current-swap 0
                    anon 4501504
                    kernel-stack 73728
                    slab 2453504
                    sock 8192
                    anon-thp 0
                    file 0
                    file-writeback 0
                    file-dirty 0
                }
                cpuacct-statistics {
                    user 34323091
                    system 12342430
                }
            }
        }
    }
```

10.10.5.1.2 Showing current OOM adjust scores

The output below is an example of showing information about the current OOM adjust scores for all applications that are managed by the app_mgr.

```
--{ running }--[  ]--
# info from state system app-management application * oom-score-adj
    system {
        app-management {
            application aaa_mgr {
                oom-score-adj 0
            }
            application acl_mgr {
                oom-score-adj 0
            }
            application app_mgr {
                oom-score-adj -998
            }
            application arp_nd_mgr {
                oom-score-adj -200
            }
|
|
|
            application vxlan_mgr {
                oom-score-adj 0
            }
            application xdp_lc_1 {
                oom-score-adj -200
            }
        }
    }
```

10.10.5.1.3 Showing cgroup information

The output below is an example of showing information about cgroups that are associated with applications managed by the app_mgr.

```
--{ running }--[  ]--
# info from state system app-management application * cgroup
    system {
        app-management {
            application aaa_mgr {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
            application acl_mgr {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
            application arp_nd_mgr {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
            application bgp_mgr {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
|
|
|
            application sshd-mgmt {
                cgroup /mnt/cgroup/unified/workload.slice/secondary
            }
            application supportd {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
            application vxlan_mgr {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
            application xdp_lc_1 {
                cgroup /mnt/cgroup/unified/workload.slice/primary
            }
        }
    }
```

10.10.5.1.4 Listing all the applications associated with a specified cgroup

Use the **tools system cgroup command pgrep cgroup** *cgroupname* command to list all the applications associated with a specified group; the output below shows an example.

```
--{ running }--[  ]--
# tools system cgroup command pgrep cgroup workload.slice/primary
+-------+----------------------+
|  Pid  |       Process        |
+=======+======================+
| 3373  | sr_app_mgr           |
| 3385  | sr_supportd          |
| 3402  | sr_device_mgr        |
| 3460  | sr_idb_server        |
| 3471  | sr_aaa_mgr           |
| 3482  | sr_acl_mgr           |
| 3518  | sr_arp_nd_mgr        |
| 3542  | sr_chassis_mgr       |
| 3560  | sr_dhcp_client_mgr   |
| 3574  | sr_evpn_mgr          |
| 3586  | sr_fib_mgr           |
```

```
| 3598  | sr_l2_mac_learn_mgr  |
| 3611  | sr_l2_mac_mgr        |
| 3621  | sr_lag_mgr           |
| 3631  | sr_linux_mgr         |
| 3641  | sr_log_mgr           |
| 3651  | sr_mcid_mgr          |
| 3681  | sr_mgmt_server       |
| 3702  | sr_net_inst_mgr      |
| 3724  | sr_oam_mgr           |
| 3743  | sr_sdk_mgr           |
| 3753  | sr_sflow_sample_mgr  |
| 3772  | sr_xdp_lc_1          |
| 3874  | sudo                 |
| 3895  | sudo                 |
| 3902  | sudo                 |
| 3921  | sudo                 |
| 3930  | sr_qos_mgr           |
| 3953  | sr_gnmi_server       |
| 3979  | sr_json_rpc          |
| 3990  | sr_vxlan_mgr         |
| 16740 | sr_json_wkr          |
| 16742 | python3              |
| 17013 | python3              |
| 18474 | sr_json_wkr          |
| 18478 | python3              |
| 18671 | python3              |
| 20671 | sr_lldp_mgr          |
| 21177 | sr_bgp_mgr           |
| 21190 | sr_plcy_mgr          |
| 12753 | sr_l2_static_mac_mgr |
| 27353 | rsyslogd             |
| 9777  | python               |
+-------+----------------------+
```

10.10.5.2 Linux-provided cgroup debugging commands

The following Linux-provided CLI commands are available for debugging cgroups:

- the **systemd-cgls** command
- the **systemd-cgtop** command

The **systemd-cgls** command dumps the cgroup hierarchy. The output below shows an example of the **systemd-cgls** command.

```
[linuxadmin@srlinux unified]$ systemd-cgls
+-1 /usr/lib/systemd/systemd --switched-root --system --deserialize 22
+-system.slice
  +-rngd.service
  | +-2725 /sbin/rngd -f
  +-systemd-udevd.service
  | +-868 /usr/lib/systemd/systemd-udevd
  +-system-serial\x2dgetty.slice
  | +-serial-getty@ttyS0.service
  |   +-1120 login -- linuxadmin
  |   +-8320 -bash
  |   +-9262 sendacct "systemd-cgls"
  |   +-9263 systemd-cgls
  |   +-9264 less
  +-srlinux.service
  | +- 3092 /usr/bin/sudo /opt/srlinux/bin/sr_linux
  | +- 3120 runuser --user srlinux --group srlinux -- /opt/srlinux/bin/sr_linux
```

```
  | +- 4019 /bin/bash /opt/srlinux/bin/sr_linux --user-env-switched
  | +- 4039 ./sr_app_mgr
  | +- 4051 ./sr_supportd --server-mode
  | +- 4068 ./sr_device_mgr
  | +- 4219 ./sr_idb_server
  | +- 4229 ./sr_eth_switch
  | +- 4288 ./sr_aaa_mgr
  | +- 4299 ./sr_acl_mgr
  | +- 4314 ./sr_arp_nd_mgr
  | +- 4324 ./sr_chassis_mgr
  | +- 4337 ./sr_dhcp_client_mgr
  | +- 4365 ./sr_evpn_mgr
  | +- 4381 ./sr_fib_mgr
  | +- 4395 ./sr_l2_mac_learn_mgr
  | +- 4411 ./sr_l2_mac_mgr
  | +- 4423 ./sr_lag_mgr
  | +- 4445 ./sr_linux_mgr
  | +- 4494 ./sr_log_mgr
  | +- 4510 ./ntpd -c /etc/ntps.conf -g
  | +- 4526 ./sr_mcid_mgr
  | +- 4574 ./sr_mgmt_server
  | +- 4608 ./sr_net_inst_mgr
  | +- 4626 ./sr_oam_mgr
  | +- 4640 ./sr_sdk_mgr
  | +- 4658 ./sr_sflow_sample_mgr
  | +- 4681 ./sr_xdp_cpm
  | +- 5197 /usr/bin/sudo -Enu root /usr/bin/sudo -Enu gnmirpc bash -c ./sr_gnmi
  | +- 5224 /usr/bin/sudo -Enu root /usr/bin/sudo -Enu jsonrpc bash -c ./sr_json
  | +- 5243 /usr/bin/sudo -Enu gnmirpc bash -c ./sr_gnmi_server
  | +- 5248 ./sr_qos_mgr
  | +- 5261 /usr/bin/sudo -Enu jsonrpc bash -c ./sr_json_rpc
  | +- 5290 ./sr_gnmi_server
  | +- 5302 ./sr_json_rpc
  | +- 5693 /usr/sbin/sshd -f /etc/ssh/sshd_config_mgmt
  | +- 6361 /usr/sbin/rsyslogd -i /var/run/srlinux/rsyslogd.pid
  | +-21936 ./sr_bfd_mgr
  | +-22018 ./sr_bgp_mgr
  | +-22132 ./sr_isis_mgr
  | +-22242 ./sr_lldp_mgr
  | +-22321 ./sr_mpls_mgr
  | +-22437 ./sr_ospf_mgr
  | +-22588 ./sr_plcy_mgr
  | +-22701 ./sr_static_route_mgr
  | +-22796 sr_json_wkr
  | +-22797 python3 -m srlinux.mgmt.json_rpc.json_rpc_server_main --bind=[::]:40
  | +-22953 ./ntpd -c /etc/mntp.conf -g
  | +-23023 python3 -m srlinux.mgmt.json_rpc.json_rpc_server_main --bind=[::]:40
  | +-23736 sr_json_wkr
  | +-23741 python3 -m srlinux.mgmt.json_rpc.json_rpc_server_main --bind=[::]:40
  | +-24009 python3 -m srlinux.mgmt.json_rpc.json_rpc_server_main --bind=[::]:40
  | +-27488 ./dnsmasq --conf-file=/etc/dnsmasq.conf
  +-polkit.service
  | +-2701 /usr/lib/polkit-1/polkitd --no-debug
  +-ztpapi.service
  | +-1114 /opt/srlinux/ztp/virtual-env/bin/python -m ztp.ztphttp
  +-systemd-journald.service
  | +-840 /usr/lib/systemd/systemd-journald
  +-sshd.service
  | +-985 /usr/sbin/sshd -D
  +-crond.service
  | +-1133 /usr/sbin/crond -n
  +-sr_watchdog.service
  | +-1155 sr_wd
  +-dbus.service
```

```
  | +-991 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile
  +-systemd-logind.service
    +-1062 /usr/lib/systemd/systemd-logind
```

The **systemd-cgtop** command dumps the current usage of each cgroup. The output below shows an example of the **systemd-cgtop** command.

```
Path                                    Tasks   %CPU   Memory  Input/s Output/s

/                                         186      -        -        -        -
/system.slice/crond.service                 1      -        -        -        -
/system.slice/dbus.service                  1      -        -        -        -
/system.slice/polkit.service                1      -        -        -        -
/system.slice/rngd.service                  1      -        -        -        -
/system.slice/sr_watchdog.service           1      -        -        -        -
/system.slice/srlinux.service              53      -        -        -        -
/system.slice/sshd.service                  1      -        -        -        -
/system.slic...ial-getty@ttyS0.service      3      -        -        -        -
/system.slice/systemd-journald.service      1      -        -        -        -
/system.slice/systemd-logind.service        1      -        -        -        -
/system.slice/systemd-udevd.service         1      -        -        -        -
/system.slice/ztpapi.service                1      -        -        -        -
```

11 UFT profiles

The processors in 7220 IXR-D1, 7220 IXR-D2/D3, and 7220-IXR H2/H3 systems use shared memory tables known as Unified Forwarding Tables (UFTs). With UFTs, the processor has a set of shared banks that can be partitioned to support the following types of forwarding table entries:

- Learned MAC addresses
- IP host entries
- IP longest-prefix-match routes

The default settings for the shared bank allocations can be changed by modifying the UFT profile for the system.

11.1 Shared bank partitioning for SR Linux systems

The number of shared banks, the size of each shared bank, and the way they can be divided depend on the SR Linux system, as described in Table 9: Shared bank configuration for SR Linux systems.

Table 9: Shared bank configuration for SR Linux systems

SR Linux system	Shared bank configuration
7220 IXR-D1	6 shared banks 16K single-wide/SW or 8K double-wide/DW entries per shared bank • 1 SW entry can store 1 IPv4 host entry or 1 MAC address. • 1 DW entry can store 1 IPv6 host entry. • If a bank has any DW entry, it is forced into DW mode. If ALPM is enabled, it requires 4 shared banks; each of the remaining banks can be allocated to either IP host entries or MAC addresses.
7220 IXR-D2/D3	8 shared banks 32K single-wide of 16K double-wide entries per shared bank • 1 SW entry can store 1 IPv4 host entry or 1 MAC address. • 1 DW entry can store 1 IPv6 host entry. • If a bank has any DW entry, it is forced into DW mode. ALPM can be enabled in 8-bank (high-scale) mode; each of the remaining banks, if there are any, can be allocated to either IP host entries or MAC addresses.
7220 IXR-H2/H3	8 shared banks 8K entries per shared bank

SR Linux system	Shared bank configuration
	ALPM is enabled by default. ALPM uses all of the shared banks.

11.2 LPM table partitioning

IP FIB scale depends significantly on the number of UFT banks used for ALPM, but also depends on the partitioning of the hardware LPM table, which is an always-present TCAM + SRAM table that stores IP LPM route entries.

When ALPM is not active, IP FIB scale is entirely determined by the size and partitioning of this table. When ALPM is active, the hardware LPM table is used as a first-search table into the ALPM banks, so it also plays an important role.

If the **ipv6-128bit-lpm-entries** parameter is configured to be greater than zero, the hardware LPM table is partitioned into two sub-blocks: a single-wide sub-block and a double-wide sub-block:

- The single-wide sub-block can store IPv4 routes (each consuming half of a single-wide entry) and IPv6 routes up to /64 prefix length (each consuming a single-wide entry).
- The size of the double-wide sub-block is controlled by the **ipv6-128bit-lpm-entries** parameter, and this sub-block can store IPv6 routes up to /128 prefix length (each consuming a double-wide entry).

  It can also store IPv4 routes and IPv6 routes up to /64 prefix length; inside the double-wide sub-block, IPv4 routes consume half of a single-wide entry, and IPv6 routes up to /64 prefix length require a double-wide entry.

11.3 Default UFT allocations for SR Linux systems

The default UFT shared bank allocations and hardware LPM table parameters for each SR Linux system are summarized in Table 10: Default UFT allocations and hardware LPM table parameters.

Table 10: Default UFT allocations and hardware LPM table parameters

SR Linux system	Default UFT allocations and L3DEFIP table parameters
7220 IXR-D1	Extra IP host shared banks: 3 Extra MAC address shared banks: 3 ALPM: disabled **ipv6-128bit-lpm-entries**: 1024
7220 IXR-D2/D3	Extra IP host shared banks: 4 Extra MAC address shared banks: 4 ALPM: disabled **ipv6-128bit-lpm-entries**: 2048
7220 IXR-H2/H3	ALPM: enabled **ipv6-128bit-lpm-entries**: 512

11.4 Configuring a UFT profile

Procedure

You can change the settings for the UFT shared bank allocations and hardware LPM table parameters from the defaults listed in Table 10: Default UFT allocations and hardware LPM table parameters by modifying the system UFT profile.

To change the settings for the UFT shared bank allocations and hardware LPM table parameters from the defaults, modify the system UFT profile as shown in the following example.

Example:

This example configures a UFT profile for a 7220 IXR-D1 system. The UFT profile enables ALPM and configures 32K extra IP host entries from the UFT shared banks.

```
--{ * candidate shared default }--[  ]--
# info platform resource-management unified-forwarding-resources
    platform {
        resource-management {
            unified-forwarding-resources {
                alpm enabled
                requested-extra-ip-host-entries 32768
                ipv6-128bit-lpm-entries 1024
            }
        }
    }
```

Note:
UFT profile configuration changes do not take effect immediately when the changes are committed; they take effect the next time XDP is restarted.

11.5 Displaying UFT profile information

Procedure

Use the **info from state** command to display the UFT profile, including the extra number of host entries and MAC address entries allocated from UFT shared banks.

Example:

```
--{ * candidate shared default }--[  ]--
# info from state platform resource-management unified-forwarding-resources
    platform {
        resource-management {
            unified-forwarding-resources {
                xdp-restart-required true
                alpm enabled
                requested-extra-ip-host-entries 32768
                allocated-extra-ip-host-entries 32768
                allocated-extra-mac-entries 0
                ipv6-128bit-lpm-entries 1024
            }
        }
    }
```

In the example, the `xdp-restart-required` leaf is shown as `true` if a change has been committed to one or more of the configurable values in the `unified-forwarding-resources` container, but XDP has not yet been restarted. Until XDP is restarted, the operational values are still the values initialized at the last XDP restart.

12 Maintenance Mode

SR Linux maintenance mode allows you to take a network element out of service so that maintenance actions can be performed; for example, to upgrade the software image on a router.

Using SR Linux maintenance mode, you can do this with minimal impact on traffic. SR Linux maintenance mode works as follows:

1. A maintenance group is configured that specifies the resources to be taken out of service. See Configuring a maintenance group.
2. A maintenance profile is configured that specifies policy changes to apply when the group is in maintenance mode. A maintenance profile is associated with each maintenance group. See Configuring a maintenance profile.

   The usual intent of the policy changes is to de-preference paths through the maintenance group so that traffic is diverted elsewhere.
3. The maintenance group is placed into maintenance mode, which applies the policies in the associated maintenance profile. See Placing a maintenance group into maintenance mode.
4. The user monitors the traffic on the interfaces in the maintenance group. When the traffic rate falls below a threshold, the user shuts down the members of the maintenance group and performs the required service. See Taking a maintenance group out of service.
5. After the service is completed, the user takes the maintenance group out of maintenance mode, which disables the policies in the associated maintenance profile, and restores traffic on the original paths. See Restoring the maintenance group to service.

12.1 Configuring a maintenance group

Procedure

A maintenance group specifies a set of network resources to be taken out of service when maintenance mode is enabled. For example, a maintenance group can consist of one or more BGP neighbors or peer groups belonging to a one or more network instances.

Specify a set of network resources in a maintenance group. The network resources in the maintenance group are taken out of service when maintenance mode is enabled.

Example:

The following example configures maintenance group mgroup1, consisting of the BGP neighbors in peer group headquarters1, which exists in the default network instance, as well as BGP neighbors in peer group headquarters2, which exists in network instance "black". In the example, maintenance group mgroup1 is associated with maintenance profile mprof1.

```
--{ candidate shared default }--[  ]--
# info system maintenance
    system {
        maintenance {
            group mgroup1 {
                maintenance-profile mprof1
```

```
                maintenance-mode {
                    admin-state disable
                }
                members {
                    bgp {
                        network-instance default {
                            peer-group [
                                headquarters1
                            ]
                        network-instance black {
                            peer-group [
                                headquarters2
                            ]
                        }
                    }
                }
            }
        }
    }
```

12.2 Configuring a maintenance profile

Procedure

A maintenance profile specifies policy changes that are applied to members of a maintenance group when maintenance mode is enabled for the group.

The following example configures a routing policy and a maintenance profile that references the routing policy.

Example:

The following example defines the routing policy drain-with-as-path-prepend:

```
--{ candidate shared default }--[  ]--
# info routing-policy policy drain-with-as-path-prepend
    routing-policy {
        policy drain-with-as-path-prepend {
            default-action {
                accept {
                    bgp {
                        as-path {
                            prepend {
                                as-number auto
                                repeat-n 3
                            }
                        }
                    }
                }
            }
        }
    }
```

The following example defines a maintenance profile that references the routing policy drain-with-as-path-prepend.

```
--{ candidate shared default }--[  ]--
# info system maintenance
    system {
```

```
        maintenance {
            profile mprof1 {
                bgp {
                    import-policy drain-with-as-path-prepend
                    export-policy drain-with-as-path-prepend
                }
            }
        }
    }
```

12.3 Placing a maintenance group into maintenance mode

Procedure

When a maintenance group is placed into maintenance mode, it applies the policies in the associated maintenance profile to the resources in the maintenance group.

To place a maintenance group into maintenance mode, change the setting for **maintenance-mode** in the group configuration to **enable**, then commit the configuration.

Example:

The following example enables maintenance mode for maintenance group mgroup1, and commits the configuration. The policies configured in the maintenance profile associated with mgroup1 are applied to the resources configured in mgroup1.

```
--{ candidate shared default }--[  ]--
# info system maintenance group mgroup1
    system {
        maintenance {
            group mgroup1 {
                maintenance-mode {
                    admin-state enable
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# commit stay
All changes have been committed. Starting new transaction.
```

12.4 Taking a maintenance group out of service

Procedure

After enabling maintenance mode for a maintenance group, monitor the traffic on the interfaces in the group. When the traffic rate drops to an acceptable level, shut down the members of the maintenance group and perform the required service.

Monitor traffic for an interface using the **info from state** command to show interface traffic statistics. When the traffic rate reaches a low-enough level, administratively disable the interface.

Example:

To monitor an interface:

```
--{ running }--[  ]--
# info from state interface ethernet-1/2 statistics
    interface ethernet-1/2 {
        statistics {
            in-octets 46969
            in-unicast-pkts 492
            in-broadcast-pkts 0
            in-multicast-pkts 34
            in-discards 0
            in-errors 0
            in-unknown-protos 0
            in-fcs-errors 0
            out-octets 46169
            out-unicast-pkts 490
            out-broadcast-pkts 1
            out-multicast-pkts 25
            out-discards 0
            out-errors 0
            carrier-transitions 1
```

To shut down the interface:

```
--{ * candidate shared default }--[  ]--
# interface ethernet-1/2 admin-state disable
# commit stay
All changes have been committed. Starting new transaction.
```

12.5 Restoring the maintenance group to service

Procedure

After performing the required maintenance operations, restore the maintenance group to service.

To restore the maintenance group to service, change the setting for the `maintenance-mode` state in the group configuration to `disable`, then commit the configuration.

Example:

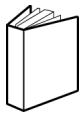
The following example takes maintenance group mgroup1 out of maintenance mode:

```
--{ candidate shared default }--[  ]--
# info system maintenance group mgroup1
    system {
        maintenance {
            group mgroup1 {
                maintenance-mode {
                    admin-state disable
                }
            }
        }
    }
```

```
--{ * candidate shared default }--[  ]--
# commit stay
```

```
All changes have been committed. Starting new transaction.
```

Customer document and product support



Customer documentation

Customer documentation welcome page

Technical support

Product support portal

Documentation feedback

Customer documentation feedback