



Nokia Service Router Linux

Release 23.10

Software Installation Guide

3HE 19831 AAAA TQZZA
Edition: 01
November 2023

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.

This document includes Nokia proprietary and confidential information, which may not be distributed or disclosed to any third parties without the prior written consent of Nokia.

This document is intended for use by Nokia's customers ("You"/"Your") in connection with a product purchased or licensed from any company within Nokia Group of Companies. Use this document as agreed. You agree to notify Nokia of any errors you may find in this document; however, should you elect to use this document for any purpose(s) for which it is not intended, You understand and warrant that any determinations You may make or actions You may take will be based upon Your independent judgment and analysis of the content of this document.

Nokia reserves the right to make changes to this document without notice. At all times, the controlling version is the one available on Nokia's site.

No part of this document may be modified.

NO WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF AVAILABILITY, ACCURACY, RELIABILITY, TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, IS MADE IN RELATION TO THE CONTENT OF THIS DOCUMENT. IN NO EVENT WILL NOKIA BE LIABLE FOR ANY DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL OR ANY LOSSES, SUCH AS BUT NOT LIMITED TO LOSS OF PROFIT, REVENUE, BUSINESS INTERRUPTION, BUSINESS OPPORTUNITY OR DATA THAT MAY ARISE FROM THE USE OF THIS DOCUMENT OR THE INFORMATION IN IT, EVEN IN THE CASE OF ERRORS IN OR OMISSIONS FROM THIS DOCUMENT OR ITS CONTENT.

Copyright and trademark: Nokia is a registered trademark of Nokia Corporation. Other product names mentioned in this document may be trademarks of their respective owners.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

© 2023 Nokia.

Table of contents

| | | |
|----------|---|-----------|
| 1 | About this guide..... | 6 |
| 1.1 | Precautionary and information messages..... | 6 |
| 1.2 | Conventions..... | 6 |
| 2 | What's new..... | 8 |
| 3 | SR Linux software overview..... | 9 |
| 3.1 | File system layout..... | 9 |
| 3.2 | Boot process..... | 11 |
| 3.3 | Secure Boot..... | 11 |
| 3.3.1 | Activate Secure Boot..... | 12 |
| 3.3.2 | Secure Boot state..... | 13 |
| 3.3.3 | Update Secure Boot variables..... | 13 |
| 4 | Deploying SR Linux container images..... | 15 |
| 4.1 | SR Linux container prerequisites..... | 15 |
| 4.2 | Launching an SR Linux container manually..... | 15 |
| 5 | Installing software..... | 20 |
| 5.1 | Hardware overview..... | 20 |
| 5.2 | Installation overview..... | 21 |
| 5.2.1 | Software image contents..... | 21 |
| 5.2.2 | Installation concepts..... | 21 |
| 5.3 | Performing software upgrades..... | 22 |
| 5.3.1 | Software upgrade using a tools command..... | 22 |
| 5.3.1.1 | Software upgrade using a HTTP/HTTPS link..... | 22 |
| 5.3.1.2 | Software upgrade using the image bin file..... | 23 |
| 5.3.2 | Software upgrade from the bash shell..... | 23 |
| 5.3.3 | In-service software upgrade..... | 26 |
| 5.3.3.1 | Minor ISSU..... | 27 |
| 5.3.3.2 | Configuration state support..... | 27 |
| 5.3.3.3 | YANG path support..... | 28 |
| 5.3.3.4 | Performing an ISSU..... | 28 |
| 5.4 | Performing recovery procedures..... | 30 |

| | | |
|----------|---|-----------|
| 5.4.1 | Creating a bootable SD card..... | 30 |
| 5.4.1.1 | SD card flash script..... | 30 |
| 5.4.1.2 | Image copy..... | 31 |
| 5.4.2 | Local rescue image..... | 32 |
| 5.5 | Bootstrapping using ONIE..... | 32 |
| 5.5.1 | Image upgrade from ONIE prompt..... | 33 |
| 5.5.2 | Installing an ONIE image..... | 35 |
| 6 | Zero Touch Provisioning..... | 37 |
| 6.1 | Applicability..... | 37 |
| 6.2 | ZTP overview..... | 37 |
| 6.2.1 | Network requirements..... | 37 |
| 6.3 | Process information..... | 39 |
| 6.3.1 | DHCP discovery and solicitation..... | 39 |
| 6.3.1.1 | Auto-provisioning options..... | 41 |
| 6.3.1.2 | DHCP server Option 42 (IPv4) and 56 (IPv6) for NTP..... | 41 |
| 6.3.2 | DHCP offer..... | 41 |
| 6.3.2.1 | Default gateway route configuration for IPv4..... | 41 |
| 6.3.2.2 | DHCP relay..... | 42 |
| 6.3.3 | Python provisioning script..... | 42 |
| 6.3.4 | Auto-provisioning failures..... | 42 |
| 6.3.5 | ZTP log files..... | 43 |
| 6.4 | Configuring ZTP..... | 43 |
| 6.4.1 | ZTP CLI versus SR Linux CLI..... | 43 |
| 6.4.2 | Configuring the Python provisioning script..... | 44 |
| 6.4.3 | Configuring the ZTP timeout value using the provisioning script..... | 46 |
| 6.4.4 | Configuring options in the grub.cfg using ZTP CLI..... | 46 |
| 6.4.5 | Managing images using ZTP CLI..... | 48 |
| 6.4.6 | Configuring the NOS using ZTP CLI..... | 50 |
| 6.4.7 | Redownloading the executable files with ZTP CLI..... | 50 |
| 6.4.8 | Starting, stopping, and restarting a ZTP process using ZTP CLI..... | 51 |
| 6.4.9 | Checking the status of a ZTP process using ZTP CLI..... | 51 |
| 6.4.10 | Configuring options in the grub.cfg using SR Linux CLI..... | 52 |
| 6.4.11 | Specifying the image, kernel, or RAM to boot the system using SR Linux CLI..... | 53 |
| 6.4.12 | Starting, stopping, and restarting a ZTP process using SR Linux CLI..... | 53 |
| 6.4.13 | Checking the status of a ZTP process using SR Linux CLI..... | 54 |

| | | |
|--|--|-----------|
| 6.5 | ZTP CLI and SR Linux CLI command structures..... | 55 |
| 6.5.1 | ZTP CLI command structure..... | 55 |
| 6.5.2 | SR Linux CLI command structure..... | 56 |
| Appendix: ZTP Python library..... | | 57 |
| | ZTPClient..... | 57 |
| | Functions..... | 57 |
| | chassis_control()..... | 58 |
| | chassis_linecards()..... | 58 |
| | configure(configurl)..... | 58 |
| | image_activate(version)..... | 59 |
| | image_bootorder(bootorder)..... | 59 |
| | image_delete(version)..... | 60 |
| | image_list()..... | 60 |
| | image_upgrade(image_url, md5_url, options)..... | 61 |
| | option_autoboot(status)..... | 61 |
| | option_bootintf(interface)..... | 62 |
| | option_clientid(type)..... | 62 |
| | option_downgrade(status)..... | 63 |
| | option_duration(timeout, retry)..... | 63 |
| | option_formatovl(status)..... | 64 |
| | option_formatsrletc(status)..... | 64 |
| | option_formatsrlopt(status)..... | 64 |
| | option_list()..... | 65 |
| | option_nosinstall(status)..... | 65 |
| | provision(provisionurl)..... | 66 |
| | service_restart()..... | 66 |
| | service_start()..... | 67 |
| | service_status()..... | 67 |
| | service_stop()..... | 67 |
| Appendix: Migrating from CentOS to Debian OS..... | | 69 |

1 About this guide

This document describes how to install the Nokia Service Router Linux (SR Linux) in various environments. It defines the required prerequisites and procedures for how to install SR Linux software elements. 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 software is installed and upgraded.



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

Configuration and command outputs shown in this guide are examples only; actual displays may differ depending on supported functionality and user configuration.

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.

In command prompt examples, # indicates a regular prompt and \$ indicates a sudo/root/privileged prompt.

2 What's new

| Topic | Location |
|-------------------------|--|
| Debian Operating system | Appendix: Migrating from CentOS to Debian OS |
| Updated procedure | Deploying SR Linux container images |
| Secure Boot | Secure Boot |

3 SR Linux software overview

This chapter describes the basic software operations on the SR Linux.

3.1 File system layout

The file system is distributed and laid out as a closed-source third-party application on the OS. The following table lists the file system layout.

Table 1: File system layout

| Path | Description |
|------------------------------|---|
| /opt/srlinux/appmgr/* | YAML configuration for Nokia-provided applications |
| /opt/srlinux/appmgr/aaa_mgr | System files for AAA manager |
| /opt/srlinux/appmgr/logmgr | System files for log manager |
| /opt/srlinux/bin/* | Application binaries |
| /opt/srlinux/firmware/* | Contains firmware (BIOS, IOCTL, CPLD, and so on) |
| /opt/srlinux/imm/imminit.tar | IMM image |
| /opt/srlinux/lib/* | Nokia-provided shared libraries |
| /opt/srlinux/models/* | Nokia-provided YANG models |
| opt/srlinux/config.json | Factory system configuration |
| /opt/srlinux/osync/* | Configuration files and exclude/include files for overlay synchronization |
| /opt/srlinux/python/* | The virtual environment used to run SR Linux Python processes |
| /opt/srlinux/systemd/* | Systemd unit files and configuration |
| /opt/srlinux/usr/* | Community-provided binaries, shared libraries, and licenses |
| /opt/srlinux/var/run/* | Running directory for process sockets |
| /opt/srlinux/ztp/* | ZTP virtual environment, templates, and client |

| Path | Description |
|------------------------------------|---|
| /etc/opt/srlinux/config.json | System configuration |
| /etc/opt/srlinux/config.json.gz | Compressed system configuration (if the configuration needs to be compressed) |
| /etc/opt/srlinux/banner | The system banner, pre-login |
| /etc/opt/srlinux/license.key | License file |
| /etc/opt/srlinux/srlinux.rc | Global environment file |
| /etc/opt/srlinux/tls/* | User-configured certificates |
| /etc/opt/srlinux/appmgr/* | YAML configuration for operator-provided agents |
| /etc/opt/srlinux/appmgr/overrides | YAML overrides for operator and Nokia-provided applications |
| /etc/opt/srlinux/models/* | YANG modules for operator-provided agents |
| /etc/opt/srlinux/checkpoint/* | Configuration checkpoints |
| /etc/opt/srlinux/devices/* | Discovered devices |
| /etc/opt/srlinux/cli/plugins/* | Operator-provided plug-ins |
| /var/opt/srlinux/run/* | Application PIDs |
| /var/log/srlinux/buffer/* | Tmpfs logging |
| /var/log/srlinux/buffer.persist/* | Persistent buffered logging |
| /var/log/srlinux/file/* | Persistent logging |
| /var/log/srlinux/debug/* | Debug logging, tmpfs |
| /var/log/srlinux/debug.persist/* | Persistent debug logging |
| /var/log/srlinux/monitor/* | Log_mgr tmpfs storage |
| /var/log/srlinux/monitor.persist/* | Log_mgr persistent storage |

| Path | Description |
|------------------------------|---|
| /var/log/srlinux/archive/* | Archive directory for previous startups |
| \$HOME/.srlinuxrc | Per-user environment |
| \$HOME/srlinux/cli/plugins/* | Per-user CLI plug-ins |

The Solid State Drive (SSD) is used for an overlay file system, allowing the user to add persistent modifications to the system.

3.2 Boot process

About this task

The SR Linux boots using a normal Linux boot mechanism. The BIOS is set up to boot from the internal storage device. The following is the normal boot sequence:

Procedure

- Step 1.** The system powers on. Assuming a fully populated system, all components initialize to the point where they bring up their link on the back door bus. Fans are under hardware control and run at 100% speed.
- Step 2.** Both control modules start their boot sequence. During this sequence, the following occurs:
 - a. The BIOS tries to boot off the internal storage device.
 - b. Grub2 loads the kernel and initramfs into memory. The initramfs contains a squashfs of the root file system used to run SR Linux, including base Debian and SR Linux applications. The squashfs is unpacked and loaded.
 - c. When the initramfs has loaded, the application manager (app_mgr) starts and loads the applications based on their start order.
 - d. The system is operational.
- Step 3.** On control-redundant platforms, both control modules attempt to boot at the same time. The control module in slot B waits up to 300 seconds before becoming active, after detecting slot 1 on the back door bus.
- Step 4.** The chassis_mgr and device_mgr initialize, push images, and boot each line card and Switch Fabric Module (SFM). This includes making any decisions based on power availability, and taking control of fans.

3.3 Secure Boot

SR Linux Secure Boot ensures that the software executed by the system is trusted and originated from Nokia IP routing.

At every boot of the control card, each step in the boot process verifies the digital signature of the next software element to boot for integrity and authenticity up to the SR Linux application contained in the squashfs root file system. This boot sequence forms the chain of trust for Secure Boot.

The Secure Boot chain is rooted in the platform control card firmware based on UEFI (Unified Extensible Firmware Interface Root of Trust) specifications. As such, Nokia Platform Key, Key Exchange Key, allowed and disallowed databases are provisioned in the system when Secure Boot is activated to perform the required signature verification.

Firmware updates are also digitally signed and verified using the same principle. The signature verification of a firmware update is performed at boot time by the existing firmware before the firmware update can proceed.

Software image signatures use RSA-4096 keys and SHA-384 hashes.

3.3.1 Activate Secure Boot

Procedure

Depending on the system, Secure Boot can be enabled from manufacturing or field enabled on compatible system control cards using CLI commands.

To activate Secure Boot on compatible control cards that do not have Secure Boot enabled from manufacturing, the following command is used providing the card slot, card serial number, and confirmation code:

```
tools platform trust secure-boot control <A|B> activate confirmation-code <string> serial-number <string>
```

The card serial number and Secure Boot confirmation code are required to avoid accidental activation of Secure Boot in the network. The confirmation code is `secure-boot-permanent`.

The Secure Boot activate command verifies that the boot chain used at the next reboot of the system is properly signed otherwise the command returns an error in CLI.



WARNING:

After Secure Boot is activated on a control card, the capability is permanently enabled and cannot be disabled. The control card permanently refuses to execute unsigned software for security reasons. As a result, it is not possible to downgrade to a software release published before the release that introduced Secure Boot for a specific platform.

Example

The following example activates Secure Boot on slot A.

```
A:srl1# tools platform trust secure-boot control A activate confirmation-code secure-boot-permanent serial-number NS22222222
```

The following example shows the warning messages and prompt returned when proceeding with Secure Boot activation:

```
WARNING: This operation will permanently activate secure boot on the card and cannot be reversed. The card will also be immediately rebooted. After activation, the system will only accept digitally signed software and will not boot using un-signed software. Are you sure you want to activate secure boot and reboot this control module (y/[n])?y
```

3.3.2 Secure Boot state

Procedure

Secure Boot status is available per control card and is obtained using the following command:

show platform trust secure-boot control <slot> detail

Example: Check Secure Boot status

The following example checks the Secure Boot on slot A.

```
A:srll# show platform trust secure-boot control A
+-----+-----+
| Slot | Operational Status |
+-----+-----+
| A    | enabled              |
+-----+-----+
```

The `Operational` status indicates whether the Secure Boot is enabled or disabled.

Example: Check detailed Secure Boot state information command per control card

The `show platform trust secure-boot control <A|B> detail` command provides detailed Secure Boot state information based on the currently used Secure Boot UEFI variables and the modification dataset such as:

- `up-to-date` status: Status of the Secure Boot variables programmed in the control module compared to the current modification dataset
- `update-required` status: Indicates which variable require updating

```
A:Dut-A# show platform trust secure-boot control A detail
```

```
=====
Show report for Secure Boot on Controller A
=====
```

Summary

Operational Status: enabled

Database Variable Modification Dataset Status

Modification Dataset Present : true

Modification Dataset Valid : true

Variables Up To Date : true

dbx Update Required : false

db Update Required : false

PK Update Required : false

KEK Update Required : false

Modification Dataset db Conflict : false

Modification Dataset dbx Conflict: false

Modification Dataset Digest :

0b5e6d97c0915ad9698634b2889da9fc37b3271d0f7914304c58f819ba037f6c

```
=====
UEFI Security Database Variables
```

3.3.3 Update Secure Boot variables

In case the Secure Boot UEFI variables are updated, specific secure boot commands are used to update the allowed (db) and disallowed (dbx) databases programmed in the CPM firmware.

To activate or update the Secure Boot, the SR Linux software image contains a modification dataset that includes the UEFI variables, which can be installed on the control cards.

If a change to the UEFI db or dbx is included in a future software release, the operator must install the new UEFI variable modification dataset before deploying the new image. The UEFI variable modification dataset is included in the binary image of the new software, in the current SR Linux file system.

Use the following command to extract and install a secure boot variable update from the specified image file:

```
tools system secure-boot install-update <file> namespace <value>
```

where,

- `file`: refers to the file path or HTTP/HTTPS/SFTP URL for the image file. Default HTTP/HTTPS URL download mode is insecure.
- `namespace`: refers to the network namespace to use when downloading the image.

For example,

```
tools system secure-boot install-update srlinux.bin
```

Use the following command to display the installed update:

```
tools system secure-boot display-update file <value>
```

where, `file` refers to the file path of the secure boot variable update package. For example,

```
tools system secure-boot display-update file /etc/opt/srlinux/secure_boot_var_update.json
```

After installing the UEFI variable modification dataset, the operator must update the UEFI databases (db/dbx) programmed in the CPM firmware.

- To update the UEFI db per control card, use the following command:

```
tools platform trust secure-boot control <A|B> update confirmation-code <string> serial-number <string>
```

- If a change to the UEFI dbx is included in the currently running software release, the operator must execute the following command per control card:

```
tools platform trust secure-boot control <A|B> revoke confirmation-code <string> serial-number <string>
```

4 Deploying SR Linux container images

This chapter describes SR Linux container installation topics. The container installation topics include:

- [SR Linux container prerequisites](#)
Ensure that prerequisites are met before launching a container.
- [Launching an SR Linux container manually](#)
Launches a single SR Linux container using a manual procedure.

4.1 SR Linux container prerequisites

Ensure that prerequisites are met before installing an SR Linux container.

Minimum system requirements:

- 4 GB RAM
- 2 v CPU
- The host machine user must have sudo privileges

Minimum software requirements:

- Docker CE installed, minimum version 18.09:
<https://docs.docker.com/get-docker/>
- SR Linux container image downloaded from the <https://github.com/nokia/srlinux-container-image> repository.



Note: Containers instantiated without a license key from Nokia are active for 14 days only.

4.2 Launching an SR Linux container manually

About this task

This procedure manually launches a single container.

Procedure

Step 1. Download the SR Linux container image from the [SR Linux container image](#) repository.

```
$ docker pull -i ghcr.io/nokia/srlinux:X.Y.Z-N.tar.xz
```

where *X* = major, *Y* = minor, *Z* = patch, and *N* = build number

Step 2. Check that the Docker image was imported correctly:

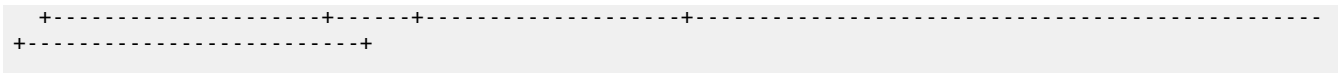
```
$ docker images
```

Example

| REPOSITORY TAG | IMAGE ID | CREATED | SIZE |
|----------------|----------|---------|------|
|----------------|----------|---------|------|

| Name | PID | State | Version |
|--------------------------|------|--------------------|-------------------------|
| Last Change | | | |
| aaa_mgr | 1209 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| acl_mgr | 1220 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| app_mgr | 1157 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.310Z | | | |
| arp_nd_mgr | 1231 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| bfd_mgr | | waiting-for-config | |
| | | | |
| bgp_mgr | 1608 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:12.022Z | | | |
| chassis_mgr | 1242 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| dev_mgr | 1178 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:09.109Z | | | |
| dhcp_client_mgr | 1253 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| dhcp_relay_mgr | | waiting-for-config | |
| | | | |
| dhcp_server_mgr | | waiting-for-config | |
| | | | |
| ethcfm_mgr | | waiting-for-config | |
| | | | |
| event_mgr | | waiting-for-config | |
| | | | |
| evpn_mgr | 1264 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| fhs_mgr | | waiting-for-config | |
| | | | |
| fib_mgr | 1275 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| gnmi_server | 2323 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:16.809Z | | | |
| gribi_server | | waiting-for-config | |
| | | | |
| idb_server | 1198 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:09.308Z | | | |
| igmp_mgr | | waiting-for-config | |
| | | | |
| isis_mgr | | waiting-for-config | |
| | | | |
| json_rpc | 2325 | running | |
| 2023-09-29T12:23:16.762Z | | | |
| l2_mac_learn_mgr | 1286 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| l2_mac_mgr | 1297 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| l2_proxy_arp_nd_mgr | | waiting-for-config | |
| | | | |
| l2_static_mac_mgr | | waiting-for-config | |
| | | | |
| label_mgr | | waiting-for-config | |
| | | | |
| lag_mgr | 1313 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| ldp_mgr | | waiting-for-config | |
| | | | |
| license_mgr | 1324 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |

| | | | |
|--------------------------|------|--------------------|--|
| linux_mgr | 1341 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| lldp_mgr | 2319 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:16.745Z | | | |
| log_mgr | 1354 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| macsec_mgr | | waiting-for-config | |
| | | | |
| mcid_mgr | 1365 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.097Z | | | |
| mfib_mgr | 1376 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| mgmt_server | 1387 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| mirror_mgr | | waiting-for-config | |
| | | | |
| mpls_mgr | | waiting-for-config | |
| | | | |
| mplsoam_mgr | 1398 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| net_inst_mgr | 1416 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| oam_mgr | | waiting-for-config | |
| | | | |
| oc_mgmt_server | | waiting-for-config | |
| | | | |
| ospf_mgr | | waiting-for-config | |
| | | | |
| p4rt_server | | waiting-for-config | |
| | | | |
| pim_mgr | | waiting-for-config | |
| | | | |
| plcy_mgr | 1617 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:12.012Z | | | |
| qos_mgr | 1626 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:12.036Z | | | |
| radius_mgr | 1434 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| sdk_mgr | 1446 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| segrt_mgr | | waiting-for-config | |
| | | | |
| sflow_sample_mgr | 1457 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |
| snmp_server-mgmt | 2405 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:18.809Z | | | |
| sshd-mgmt | 1879 | running | OpenSSH_8.0p1, OpenSSL 1.1.1n FIPS 15 Mar 2022 |
| 2023-09-29T12:23:13.690Z | | | |
| static_route_mgr | | waiting-for-config | |
| | | | |
| supportd | 511 | running | |
| 2023-09-29T12:23:08.652Z | | | |
| te_mgr | | waiting-for-config | |
| | | | |
| tepolicy_mgr | | waiting-for-config | |
| | | | |
| twamp_mgr | | waiting-for-config | |
| | | | |
| vrrp_mgr | | waiting-for-config | |
| | | | |
| vxlan_mgr | 1635 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:12.071Z | | | |
| xdp_lc_1 | 1472 | running | v23.7.1-163-gd408df6a0c |
| 2023-09-29T12:23:10.098Z | | | |



5 Installing software

This chapter describes software installation tasks. Software installation topics include:

- [Hardware overview](#)
Describes the chassis variants of each system type. Software installations can be performed on each chassis variant.
- [Installation overview](#)
Describes concepts to be familiar with before installing or upgrading.
- [Performing software upgrades](#)
Upgrades the SR Linux software on 7250 IXR, 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H systems.
- [Performing recovery procedures](#)
Installs the SR Linux software on 7250 IXR systems.
- [Bootstrapping using ONIE](#)
Installs the SR Linux software on 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H systems.

5.1 Hardware overview

SR Linux can be installed on the 7250 IXR, 7220 IXR-D, 7220 IXR-DL, and 7220 IXR-H series systems. There are multiple chassis variants of each system type. In the sections that follow, installation procedures reference the system types collectively. Software installations can be performed on each chassis variant.

The following systems are referred to collectively as 7250 IXR systems:

- 7250 IXR-6
- 7250 IXR-6e
- 7250 IXR-10
- 7250 IXR-10e

The following systems are referred to collectively as 7220 IXR-D systems:

- 7220 IXR-D1
- 7220 IXR-D2
- 7220 IXR-D3
- 7220 IXR-D4
- 7220 IXR-D5

The following systems are referred to collectively as 7220 IXR-DL systems:

- 7220 IXR-D2L
- 7220 IXR-D3L

The following systems are referred to collectively as 7220 IXR-H systems:

- 7220 IXR-H2
- 7220 IXR-H3

- 7220 IXR-H4

For information about each chassis, see the *SR Linux Product Overview*.

Each router series also has a dedicated installation guide containing complete specifications, recommendations for preparing the installation site, and procedures to install and ground the routers. See the respective chassis installation guides listed in the *SR Linux Product Overview* for more information.

5.2 Installation overview

SR Linux can be installed on the 7250 IXR, 7220 IXR-D, 7220 IXR-DL, and 7220 IXR-H series systems.

Installations can be completed using the CLI. To perform either an initial imaging, reinstallation, or an upgrade or downgrade of a system, the operation requires pushing the new image to the device, changing the boot configuration, and rebooting.

In the installation procedure examples, commands preceded by **\$** require root privilege. Commands preceded by **#** should be executed from a bash shell.

The basic installation actions performed on the system do not change, regardless of the method used to install the SR Linux (either using the CLI or manually), but the CLI method is dependent on having a working system whereas the manual method is not.

5.2.1 Software image contents

The software image is a set of files provided as part of an SR Linux distribution. The files contained in an image are:

| | |
|------------------------------|--|
| squashfs | Contains the SR Linux root file system, including any needed binaries for system operation. |
| initramfs (or initrd) | Contains an initial file system that is used to make the hardware operational before unpacking the SR Linux squashfs into memory, then switching the root file system to it. |
| kernel (or vmlinuz) | The Linux kernel is the initial program executed by the boot loader. The kernel handles all interactions between the OS and hardware. |

To perform an installation, you must have an SR Linux image, which is a bin containing these files, along with some other files used for operations and maintenance (for example, YANG models and SNMP MIBs).

5.2.2 Installation concepts

On a 7250 IXR system, SR Linux boots from an internal SD card. On a 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H system, SR Linux boots from an internal SSD. No other boot devices may be used with the system.

The internal SD or SSD contains:

- an MBR (containing the Grub2 boot loader)
- a partition used for SR Linux images
- two overlay partitions used for persistent storage

Installations can be performed manually without using the CLI. The process may also require partitioning an SD card external to the system, installing Grub into the MBR of the card, and copying the SR Linux image to the device. Use of the manual method requires advanced knowledge of Linux commands, including disk formatting, copying files, unpacking compressed images, and editing of text files. Basic knowledge of editing text files in Linux is mandatory. The manual method requires a Linux server, with an empty SD card mounted (or use of a USB-SD card adapter).

5.3 Performing software upgrades

This section describes methods to upgrade the software using the CLI. They require a working system, with SR Linux operational and the CLI available.

Software upgrade options include:

- [Software upgrade using a tools command](#)
Upgrades and deploys the software using the **tools system deploy-image** command. This method is supported on all 7250 IXR, 7220 IXR-D, 7220 IXR-DL, and 7220 IXR-H systems.
- [Software upgrade from the bash shell](#)
Upgrades the software from the bash shell using the CLI. This method is supported on all 7250 IXR, 7220 IXR-D, 7220 IXR-DL, and 7220 IXR-H systems.
- [In-service software upgrade](#)
Upgrades software across maintenance releases within the same major release (in conjunction with a warm reboot). This method is supported on 7220 IXR-D2 and D3 systems and 7220 IXR-D2L and D3L systems only.

5.3.1 Software upgrade using a tools command

You can upgrade and deploy a new software image using the **tools system deploy-image** command in the CLI. With this command, there are two methods you can use to deploy an image. You can choose to deploy using an HTTP/HTTPS link to the software, or you can copy the image bin file onto the system, then deploy it.

Run the **tools system deploy-image** command with or without a **reboot** option. The **reboot** option deploys the image and reboots the system automatically to bring up the specified image. If the **reboot** option is not added, the image is only deployed. To then perform the upgrade, the system must be rebooted separately using the **tools platform chassis reboot** command.

5.3.1.1 Software upgrade using a HTTP/HTTPS link

About this task

Deploy the image using an HTTP/HTTPS link to the software image.

The image download is insecure by default with cert verification disabled. Use the **tools system deploy-image <http link to bin file> reboot** command, where the *<http link to bin file>* links to the image.

Following the upgrade, the upgraded configuration is not saved automatically to be the startup configuration. See the "Configuration upgrades" section in the *SR Linux Configuration Basics Guide* for information about how to persist the upgraded configuration to disk.

Example: Upgrade using a HTTP/HTTPS link

In the following example, the **reboot** option is not used. After the image is deployed, the system must be rebooted separately.

```
# tools system deploy-image https://username:password@example.com/repository/srlinux-os/
21.3.0/srlinux-21.3.0-384.bin
Downloading with the srbase-mgmt namespace. Connection timeout: 5 seconds
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100 782M    0 782M    0     0  99.9M      0  --:--:--  0:00:07  --:--:-- 101M
Deploying SRL image version 21.3.0-384
2021:03:15 21:51:46:10 | EVENT | Version of new image 21.3.0-384
2021:03:15 21:51:46:10 | EVENT | Current version: 21.3.0-377, New version: 21.3.0-384
2021:03:15 21:52:21:10 | EVENT | Invoked sync call. It may take few seconds to complete.
2021:03:15 21:52:30:15 | EVENT | Syncing image with standby
Successfully deployed SRL image version 21.3.0-384
```

5.3.1.2 Software upgrade using the image bin file

About this task

Copy the image bin file onto the system, then use the **deploy-image** command after the bin file is uploaded.

Use the `tools system deploy-image <absolute path to bin file> reboot` command, where *<absolute path to bin file>* specifies the location of the bin.

Following the upgrade, the upgraded configuration is not saved automatically to be the startup configuration. See the "Configuration upgrades" section in the *SR Linux Configuration Basics Guide* for information about how to persist the upgraded configuration to disk.

Example: Upgrade using a bin file location

In the following example, the **reboot** option is used. After the image is deployed, the system reboots automatically to bring up the image.

```
# tools system deploy-image /tmp/srlinux-21.3.0-382.bin reboot
Deploying SRL image version 21.3.0-382
2021:03:16 21:08:17:57 | EVENT | Version of new image 21.3.0-382
2021:03:16 21:08:17:58 | EVENT | Current version: 21.3.0-388, New version: 21.3.0-382
2021:03:16 21:08:53:77 | EVENT | Invoked sync call. It may take few seconds to complete.
2021:03:16 21:09:01:73 | EVENT | Syncing image with standby
Successfully deployed SRL image version 21.3.0-382
--{ candidate shared default }--[ ]--
# 2021:03:16 21:10:27:17 | EVENT | Linux sync call executing
2021:03:16 21:10:27:21 | EVENT | Umount /dev/sdb1
2021:03:16 21:10:27:23 | EVENT | Umount /dev/sdb2
2021:03:16 21:10:27:26 | EVENT | sr_cli chassis reboot force requested.
21-03-16 14:10:28.472 sr_device_mgr: Chassis reboot force requested - rebooting
21-03-16 14:10:36.079 sr_device_mgr: Rebooting - chassis reboot requested
```

5.3.2 Software upgrade from the bash shell

About this task

This procedure upgrades the software from the bash shell using the CLI.

Procedure

Step 1. Copy the SR Linux image to a location that the system being installed has access to, either to a USB or SD card, or somewhere on the network (assuming that the system being installed has access to the server). Enter:

```
# cp <path-to-srlinux-image-bin> <destination-directory>
```

Example

```
# cp SRLinux-21.3.0-459.bin /mnt/removable
```

Step 2. Log in to the system being upgraded:

```
# ssh <user>@<address>
```

Example

```
# ssh linuxadmin@192.168.0.1
```

Step 3. Enter the login credentials (when prompted by the system):

- username: linuxadmin
- password: NokiaSr11!

Step 4. Copy the image to the system. Do either of the following:

- If not using removable media (USB or SD card), copy the image to the system across the network:

```
# sudo ns_exec srbase-mgmt bash
```

```
# sudo scp <user>@<server-with-srlinux-image>:<path-to-srlinux-image-bin> <local-destination>
```

Example:

```
# sudo ns_exec srbase-mgmt bash
# sudo scp serveruser@192.168.0.2:srlinux-21.3.0-459.bin /local-destination
```

- If using removable media (USB or SD card), insert either the USB or SD card into the system and mount it to a temporary directory:

```
# sudo mkdir -p /mnt/removable
```

```
# sudo mount <path-to-disk> /mnt/removable
```

Example:

```
# sudo mkdir -p /mnt/removable
# sudo mount /dev/sdc1 /mnt/removable
```

Step 5. Unpack the SR Linux image to a location that the system being installed has access to, either across the network, or to a USB or SD card that may be inserted into the active control plane module:


```
# sudo mkdir -p /mnt/nokiaos/<version>
# sudo cp <local-destination>/<srlinux-image-file.bin> /tmp/<srlinux-image-file.bin>
# sudo chmod +x /<tmp>/<srlinux-image-file.bin>
# sudo /tmp/<srlinux-image-file.bin> --target /mnt/nokiaos/<version> --noexec
```

Example

```
# sudo mkdir -p /mnt/nokiaos/21.3.0-459
# sudo cp /mnt/removable/srlinux-21.3.0-459.bin /tmp/srlinux-21.3.0-459.bin
# sudo chmod +x /tmp/srlinux-21.3.0-459.bin
# sudo /tmp/srlinux-21.3.0-459.bin --target /mnt/nokiaos/21.3.0-459 --noexec
```

- Step 6.** Start an SR Linux CLI session, and retrieve the current version of the software. Multiple images can be shown.

```
# info from state system boot image
```

Example

```
# sr_cli
# info from state system boot image
system {
  boot {
    image [
      21.3.0-449
      20.6.1-10
    ]
  }
}
```



Note: The **info from state system boot image** output only lists images present in the grub.cfg file. The **tools system boot available-images** output lists all of the images present in the system.

- Step 7.** Print the list of images in the /mnt/nokiaos directory.

```
# tools system boot available-images
```

Example

```
# sr_cli
# tools system boot available-images
['21.3.0-449*', '20.6.1-10', '21.3.0-459']
```

- Step 8.** Update the boot image list by reordering the current version behind the new version:

```
# tools system boot image [ <version> <old-version> ]
```

Example

```
# tools system boot image [ 21.3.0-459 21.3.0-449 20.6.1-10 ]
Boot order is updated with ['21.3.0-459', '21.3.0-449', '20.6.1-10']
```

- Step 9.** Reboot the chassis:

```
# tools platform chassis reboot
```

Step 10. Wait ten minutes, then log in to the device via SSH or console, and confirm the new version.

Step 11. Following the upgrade, the upgraded configuration is not saved automatically to be the startup configuration. Enter the `save startup` command to save the configuration as the startup configuration.



Note: See the "Configuration upgrades" section in the *SR Linux Configuration Basics Guide* for information about how to save new configuration upgrades.

Step 12. To avoid stale images in the `/tmp` location, Nokia recommends that you remove the `.bin` file manually after the system has successfully rebooted. The following example removes the `/tmp/srlinux-21.3.0-459.bin` file.

Example

```
# rm -rf /tmp/srlinux-21.3.0-459.bin
```

5.3.3 In-service software upgrade

This section describes the In-Service Software Upgrade (ISSU) procedures you can use to upgrade the following systems:

- 7220 IXR-D2
- 7220 IXR-D3
- 7220 IXR-D2L
- 7220 IXR-D3L



Note: An ISSU cannot be used to perform a software downgrade.

Depending on the release version, you can perform either a minor ISSU or major ISSU. A minor ISSU updates software across maintenance releases within the same major release. A major ISSU updates software across major releases within the same release year.

To perform an ISSU, the new target software image is identified, then the upgrade is performed in conjunction with a warm reboot to restart the system. During an ISSU upgrade, SR Linux maintains non-stop forwarding. A warm reboot brings down the control and management planes while the NOS reboots, and graceful restart helpers assist with maintaining the forwarding state in peers. Any control plane or management plane functions are unavailable during a warm reboot, including the refreshing of neighbors, responding to ARP/ND, and any other slow path functions.

Warm reboot leverages control plane functionality to allow remote peers to continue forwarding based on the previously learned state. This process is known as graceful restart, where the remote system is the graceful restart helper, and SR Linux, when undergoing warm reboot, is being helped.

At a high level, the ISSU process requires the following steps. For a detailed ISSU procedure, see [Performing an ISSU](#).

1. (Recommended) Back up the existing configuration.
2. Deploy the supported ISSU image using the `tools system deploy-image` command.
3. Update the first image in the leaf-list with a supported ISSU image by setting the tools model: `tools system boot image`.

4. Ensure the running configuration is saved as the startup configuration.
5. Perform a **reboot warm** (with or without **force**).

5.3.3.1 Minor ISSU

Beginning in Release R21.6.1, you can perform a minor ISSU to update software across maintenance releases within the same major release. The upgrade does not require a datapath outage. For example, you can perform a minor ISSU in the following minor release versions. The upgrade can only be to a later version of the same minor release (when the later release becomes available).

Table 2: Minor ISSU

| Minor release | Upgrade to | Examples |
|---------------|------------|-------------------------------|
| R21.6.1 | R21.6.x | R21.6.2, R21.6.3, and so on |
| R21.11.1 | R21.11.x | R21.11.2, R21.11.3, and so on |
| R22.3.1 | R22.3.x | R22.3.2, R22.3.3, and so on |
| R22.6.1 | R22.6.x | R22.6.2, R22.6.3, and so on |
| R22.11.1 | R22.11.x | R22.11.2, R22.11.3, and so on |
| R23.3 | R23.3.x | R23.3.2, R23.3.3, and so on |
| R23.7 | R23.7.x | R23.7.2, R23.7.3, and so on |

5.3.3.2 Configuration state support

Before performing a warm reboot as the final step of an ISSU, you can first confirm if the current SR Linux configuration and state supports warm reboot (including any destination image checks for ISSU). Use the **tools platform chassis reboot warm validate** command.

```
--{ running }--[ ]--
A:# tools platform chassis reboot warm validate
/platform:
  Warm reboot validate requested

/:
  Success
```

If the validation is successful, proceed with the warm reboot.

If a validation is unsuccessful, or if an attempt to perform a warm reboot fails, you can force the warm reboot using the additional **force** option. A warm reboot may not be successful if, for example, a peer does not support graceful restart. Force the warm reboot using the **tools platform chassis reboot warm force** command.

An unsuccessful validation or a failed warm reboot attempt cannot be forced using the additional **force** option in the following cases:

- The running configuration contains configuration paths that are not supported in ISSU. To complete the ISSU, invalid configuration paths must be removed from the running configuration. See [YANG path support](#).

- The running configuration has not been saved as startup configuration.



Caution: Forcing a warm reboot may result in a service outage. The **force** option overrides any warnings, such as peers that are not configured, or peers that do not support graceful restart.

```
--{ running }--[ ]--
A:# tools platform chassis reboot warm force
/platform:
  Warm reboot force requested

/:
  Success
```

5.3.3.3 YANG path support

The following YANG paths must exist in your configuration or via inheritance (if their context is present) for a warm reboot to succeed without an outage:

```
network-instance protocols bgp graceful-restart warm-reboot admin-state enable
network-instance protocols bgp group graceful-restart warm-reboot admin-state enable
network-instance protocols bgp neighbor graceful-restart warm-reboot admin-state enable
```

The following YANG paths must not exist in a configuration for a warm reboot to succeed without an outage:

```
interface hold-time
interface lag
interface sflow
interface subinterface local-mirror-destination
interface subinterface ipv4 arp evpn
interface subinterface ipv6 neighbor-discovery evpn
interface subinterface type local-mirror-dest
network-instance protocols bgp evpn
network-instance protocols bgp group evpn
network-instance protocols bgp neighbor evpn
network-instance protocols bgp-evpn
network-instance protocols isis
network-instance protocols ospf
network-instance vxlan-interface
platform resource-management unified-forwarding-resources
system mirroring
system network-instance protocols evpn
system sflow
tunnel-interface
```

5.3.3.4 Performing an ISSU

About this task

You can perform an ISSU on 7220 IXR-D2 or D3 systems only. Instead of rebooting the chassis to bring up the new software image, you will perform a warm reboot to conclude the upgrade. During the warm reboot, the system maintains non-stop forwarding.

The warm reboot process requires a minimum 100 MB of free disk space in the file system under `/etc/opt/sr/linux`, excluding the files under the `warmboot/` directory. If the disk space is unavailable, the warm reboot will fail.

The examples in this section show an ISSU from SR Linux R21.6.1 to the next available maintenance release.



Note: When the control plane goes down during an ISSU, all SSH sessions are disconnected. Nokia recommends that you perform ISSU via a console session.



Note: Before you perform an ISSU, Nokia recommends you back up your existing configuration.

You can perform an ISSU upgrade in conjunction with the **tools system deploy-image** command. With this command, you can choose between two methods to deploy an image; you can choose to deploy using an HTTP/HTTPS link to the software, or you can copy the image bin file onto the system, then deploy it.

Procedure

Step 1. Using one of the methods described in [Software upgrade using a tools command](#), deploy the new software image with the **deploy-image** command.

Step 2. Warm reboot the chassis to begin the upgrade. During the ISSU, the system maintains non-stop forwarding. The control plane goes down.

```
# tools platform chassis reboot warm
```

Example

```
--{ running }--[ ]--
A:# tools platform chassis reboot warm
/platform:
  Warm reboot requested

/:
  Success

--{ running }--[ ]--
A:#
--{ [WARM BOOT] [FACTORY] running }--[ ]--
```

Step 3. The control plane comes back up and the SR Linux CLI is available again. Note the `[WARM BOOT]` indicator is still present in the banner as the upgrade is not yet complete.

Example

```
A:#
--{ [WARM BOOT] [FACTORY] running }--[ ]--
```

Step 4. When the warm reboot finishes, the ISSU is complete. The system will accept new configurations. The `[WARM BOOT]` indicator is no longer present in the banner.

Example

```
A:#
--{ running }--[ ]--
A:#
Current mode: running
```

Step 5. Optionally, you can use the **show version** command to confirm the new software image is running.

Example

```
A:# show version
Hostname           :
Chassis Type      : 7220 IXR-D2
Part Number       : Sim Part No.
Serial Number     : Sim Serial No.
System MAC Address: 00:01:01:FF:00:00
Software Version  : v21.6.2-384
Build Number      : 28570-g51d538796f
Architecture     : x86_64
Last Booted      : 2021-06-22T09:08:58.762Z
Total Memory     : 64151761 kB
Free Memory      : 52237679 kB
```

5.4 Performing recovery procedures

This section describes recovery procedures applicable to 7250 IXR systems.

5.4.1 Creating a bootable SD card

Installing the software requires a working Linux system running Debian 12, with access to an SD card (preferably 16 GB). A USB adapter may be used, as most servers do not have SD card slots. The SD card should be formatted and have no important data present on it. Any data on the card is wiped during the procedure. Installing the software manually requires downloading a script. In the following examples, /dev/sdb is used as the SD card device, and all steps should be completed as a user with root privileges.

This section describes methods to create a bootable SD card containing the SR Linux software image to use on a 7250 IXR system.

5.4.1.1 SD card flash script

About this task

Using a Linux machine (running Debian 12), you can install the SR Linux image on a 7250 IXR system using a flash script.



WARNING: If used incorrectly, this procedure could be destructive and may render the system creating the SD card inoperable. Verify the correct drive is being used before completing the installation.

Procedure

Step 1. Copy the SR Linux image and SR Linux rescue image to either an SD card or USB drive and insert it into the system. Alternatively, copy the images to the server being used to prepare the SD card. Use the following commands:

```
# cp <path-to-srlinux-image.bin> <destination-directory>
```

Example

```
# cp /mnt/removable/SRLinux-21.3.0-459.bin /tmp
```

Step 2. Wipe the SD card and ensure that you correctly identify the SD card, as this action is destructive.

Step 3. Download and install the following packages on the system.

```
# sudo apt install e4fsprogs
# sudo apt install grub2
# sudo apt install grub2-efi-x64.x86_64
# sudo apt install grub2-efi-x64-modules
```

Step 4. Upgrade mkfs.fat to version 4.1 or later.

Example

```
# wget http://ftp.de.debian.org/debian/pool/main/d/dosfstools/dosfstools_4.2-1_
amd64.deb
# sudo apt-get install dosfstools_4.2-1_amd64.deb
```

Step 5. Download the sdcardflash.sh script.

Step 6. Run the script.

Example

```
# /tmp/sdcardflash.sh -v -e 21.3.0-459 -i srlinux-21.3.0-459.bin -s /dev/sdb -g
"autoboot nosinstall" -m
```

Step 7. Physically remove the SD card from the system.

Step 8. Repeat steps 2 to 7 with another SD card for the standby control plane module (if applicable).

Step 9. Remove both control plane modules from the system (see either the *SR Linux 7250 IXR-6 and IXR-10 Chassis Installation Guide* or *SR Linux 7250 IXR-6e and IXR-10e Chassis Installation Guide* for a procedure), then insert the SD cards into the internal SD slot for each module.

Step 10. Insert the control plane modules into the chassis, and power the chassis on.

5.4.1.2 Image copy

About this task

Using a Linux machine (running Debian 12), you can copy an SR Linux image from one SD card to another SD card. You can then use this second SD card to install the SR Linux software image onto a 7250 IXR system.

Procedure

Step 1. Insert an SD card containing an SR Linux image into any Linux machine with a supporting SD slot.

In this procedure example, the SD card device is detected as `/dev/sdb`.

Step 2. When the SD card is detected, copy the SR Linux image to the Linux machine:

```
sudo dd if=/dev/sdb of=sd.img
```

Step 3. Remove the SD card from the Linux machine.

- Step 4.** Insert the second SD card (to which the image is copied) into the Linux machine.
- Step 5.** Copy the SR Linux image from the Linux machine to the second SD card:
`sudo dd if=sd.img of=/dev/sdb`
- Step 6.** Remove the second SD card from the Linux machine.
Insert this SD card into the internal SD card slot of a 7250 IXR system's control plane module.
The system powers on with the image.

5.4.2 Local rescue image

About this task

From a 7250 IXR system running SR Linux, you can create a bootable SD card locally on the DUT, which can then be transferred and used in another system.

Procedure

- Step 1.** Log in to the system running SR Linux via a console connection.
- Step 2.** Reboot the system and select `srlinux-rescue` from the image boot menu.
- Step 3.** Copy the `srlinux-xxx.bin` file using SCP.
To allow this, one of the ports on the system in the rescue image should have management connectivity. If there is no IP assigned to any of the ports automatically, you can add an IP manually using the `ifconfig` command:

```
ifconfig <port> <ip address> netmask <ip address>
```

Example

```
ifconfig eth4 192.168.255.254 netmask 255.255.255.0
```

- Step 4.** Find the target SD card device.
In this procedure example, the SD card device is detected as `/dev/sdb`.
- Step 5.** Find the current internal SD card device.
In this procedure example, the SD card device partition is detected as `/dev/sdc1`.
- Step 6.** Run the following command to flash the SD card device with the `srlinux-xxx.bin` file:
`bash <srlinux-xxx.bin> -- --dev <target SD device> --no-onie --source-efi <internal SD device>`

Example

```
bash srlinux-21.3.0-459.bin -- --dev /dev/sdb --no-onie --source-efi /dev/sdc1
```

- Step 7.** Remove the SD card from the system. Insert it into the internal SD card slot on the control plane module of the system where the software image is to be installed.
- Step 8.** Power on the system with the new image.

5.5 Bootstrapping using ONIE

This section describes ONIE installation procedures applicable to 7220 IXR-D, 7220 IXR-DL, and 7220 IXR-H systems.

5.5.1 Image upgrade from ONIE prompt

About this task

If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrapping to retrieve the image.



Note: ZTP install is not supported when SR Linux services are enabled on the system. If you change back to the ZTP installation method from manual bootstrapping, you must enter the following commands:

```
systemctl enable ztp /opt/srlinux/systemd/ztp.service
systemctl disable /opt/srlinux/systemd/srlinux.service
```

Procedure

- Step 1.** In the GRUB selection screen, select ONIE.
- Step 2.** From the list of ONIE boot options, select ONIE: OS Install mode.
- Step 3.** After the ONIE image boots, the service discovery starts automatically. To stop the service discovery, execute:
ONIE:/ # onie-stop
- Step 4.** Configure the management IP address and the default route to copy the SR Linux image to the 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H:

Example

```
ONIE:/ #
ONIE:/ # ifconfig eth0 135.227.251.182 netmask 255.255.255.0
ONIE:/ # ip route add 0.0.0.0/0 via 135.227.248.1
IP: RTNETLINK answers: Network is unreachable
ONIE:/ #
```

- Step 5.** Using the **SCP** command, copy the SR Linux image <version>.bin to the root folder. The "root" user password field is blank.
- Step 6.** To install SR Linux, execute the following command:
onie-nos-install <bin>

Example

```
ONIE:/ # onie-nos-install /root/srlinux-20.6.1-21398.bin
discover: installed mode detected.
Stopping: discover... done.
ONIE: Executing installer: /root/srlinux-20.6.1-21398.bin
/dev/console
Verifying archive integrity... 100% MD5 checksums are OK. All good.
Uncompressing srlinux-20.6.1-21398 100%
Files used: srlinux-20.6.1-21398.squashfs, initramfs-4.18.39-2.x86_64-02.img, vmlinuz-4.19.39-2.x86_64
```

```

Found ONIE-BOOT on /dev/sda2
Will use /dev/sda as install dev
Parts used: old_part_start[4], efi_part[4], nos_part[5], etc_part[6], opt_part[7],
data_part[8]
Remove existing partitions from /dev/sda
/dev/sda4 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.

```

Step 7. After the image is installed, the 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H reboots with the SR Linux image:

Example

```

Starting Wait for Plymouth Boot Screen to Quit...
Starting Terminate Plymouth Boot Screen...
[ OK ] Started Login Service.

SRLINUX 20.6.1-21398
Kernel 4.19.39-2.x86_64 on an x86_64

Localhost login: linuxadmin
Password: 2020:06:21 19:52:54:54 | EVENT | Starting ZTP process

[linuxadmin@localhost ~]$ system2020:06:21 19:52:58:64 | EVENT | Set link mgmt0 up
ctl disable z2020:06:21 19:53:03:82 | EVENT | ZTP Perform DHCP_V4. attempt[1]
t2020:06:21 19:53:04:14 | EVENT | Received dhcp lease on mgmt0 for 135.227.251.182/21
2020:06:21 19:53:04:23 | EVENT | option 66 provided by dhcp: http://135.277.248.118
2020:06:21 19:53:04:23 | EVENT | option 67 provided by dhcp: duts/SD-RD2-126/ztp-
config.yml

```

Step 8. Enter the login credentials:

- username: linuxadmin
- password: NokiaSr11!

Step 9. Disable the ZTP autoboot using the following command:

```
ztp service stop --autoboot disable
```

Example

```

$ ztp service stop --autoboot disable
Warning: This command to be used under guidance of Nokia Technical Support only.
Unsupported usage can trigger instability of network
+-----+-----+
| key   | value |
+-----+-----+
| status | Service stopped |
+-----+-----+
[linuxadmin@localhost ~]$

```

Step 10. After a reboot, SR Linux service is started automatically.

Step 11. If the ZTP server is set up, the mgmt0 IP address is automatically configured by the DHCP server since configuration for that is enabled by default.

Example

```

# info interface mgmt0
interface mgmt0 {
    admin-state enable
}

```

```

subinterface 0 {
  admin-state enable
  ipv4 {
    admin-state enable
    dhcp-client {
    }
  }
  ipv6 {
    admin-state enable
    dhcp-client {
    }
  }
}
}

```



Note: To manually configure the mgmt0 IP via CLI, you can add an IP address for it after deleting the `dhcp-client` (highlighted in bold) under the subinterfaces (`interface.subinterface.ipv4` and `interface.subinterface.ipv6`). Default route can be added using static route, see [Configuring static routes](#).

5.5.2 Installing an ONIE image

About this task

Installing an ONIE image on a 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H system requires a working Linux system and a USB device. Installation also requires the ONIE boot loader install environment.



WARNING: Installing the ONIE from the USB wipes out all SSD partitions.

Procedure

- Step 1.** Request the ONIE recovery .iso image for the respective 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H system from OLCS.
- Step 2.** Copy the ONIE recovery .iso image file to a USB using the following command:
`dd if=<machine>.iso of=/dev/sdX bs=10M`
 where *machine* = the image name for the device and *sdX* = the USB device name.
- Step 3.** After the ONIE recovery .iso image is copied, unmount the USB device and remove it from the Linux machine.
- Step 4.** Insert the USB into the 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H system and power the system on.
- Step 5.** When the setup message comes up, press either the **DEL** or **ESC** key to enter the BIOS interface:

```

Version 2.19.1266. Copyright (C) 2019 American Megatrends, Inc.
BIOS Date: 11/01/2019 15:48:23 Ver: 0ACHI037 Minor_Ver: V1.03
Press <DEL> or <ESC> to enter setup.
Entering Setup...

```

- Step 6.** In the BIOS prompt, select **Boot Device as USB**, then **Save & Exit**.
- Step 7.** Install the ONIE from the USB. Select **ONIE: Embed ONIE** in the GNU Grub screen.

- Step 8.** After the ONIE installation is complete, remove the USB to boot the ONIE from the SSD.
- Step 9.** After the device boots the ONIE from the SSD, select **ONIE: Install OS** in the GNU Grub screen.
- Step 10.** Verify the platform, version, and build date of the installed ONIE image:

```
GRUB loading.
Welcome to GRUB!

Platform : x86_64-nokia_ixr7220_d3-r0
Version  : 2019.02-onie_version-v1.5
Build Date: 2020-02-13T15:05+08:00

telnet>
```

- Step 11.** The device boots and enters the ONIE :/ # prompt.

The ONIE service discovery automatically gets a device IP address from a ZTP server, and the SR Linux image is downloaded.



Note: If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrap procedure to complete the installation. See the [Image upgrade from ONIE prompt](#) procedure to continue.

- Step 12.** After the SR Linux software installation completes, the 7220 IXR-D, 7220 IXR-DL, or 7220 IXR-H reboots with the updated SR Linux image. The SR Linux services and applications are automatically started.

6 Zero Touch Provisioning

This chapter describes Zero Touch Provisioning (ZTP) on SR Linux. Traditional deployment of new nodes in a network is a multistep process where the user has to connect to the hardware and provision global and local parameters.

ZTP automatically configures the nodes by obtaining the required information from the network and provisioning them with minimal manual intervention and configuration. The technician installs the nodes into the rack and when power is applied, and if connectivity is available, the nodes are auto-provisioned.

6.1 Applicability

The following implementation is currently supported:

- auto-boot using the Out-of-Band (OOB) port, which includes support of HTTP, HTTPS, TFTP, and FTP



Note: No VLANs are supported on the management port.

- dual-stack IPv4 and IPv6 (including DHCP client IPv4/IPv6)

6.2 ZTP overview

For auto-boot using the OOB port, the node storage device ships with the SR Linux image and the `grub.cfg`. Within the `grub.cfg` is an auto-boot flag. The auto-boot flag is enabled by default and can be manually changed if needed.

When the SR Linux boots, it checks the `grub.cfg` for the auto-boot flag. As the flag is set by default, the node enters auto-boot mode.

When initiated, the auto-boot mode starts the auto-provisioning process. The auto-provisioning process discovers the IP address of the node and provisions the node based on a Python provisioning script.

The DHCP server provides the node with the location of the provisioning script using Option 66 and 67, or Option 43. The node uses this URL to download the provisioning script. The provisioning script contains the location of the RPMs, configurations, images, and scripts. These files are downloaded to the storage device.

During provisioning, all events are logged and displayed at the console for debugging. After the process completes, the auto-boot flag is removed from the `grub.cfg` file. This ensures that after a successful auto-boot, additional reboots of the node do not enter auto-boot mode.

6.2.1 Network requirements

ZTP requires the following components:

- DHCP server (IPv4 or IPv6) – To support the assignment of IP addresses through DHCP requests and offers.
- file server – For staging and transfer of RPMs, configurations, images, and scripts. HTTP, HTTPS, TFTP, and FTP are supported. For HTTPS, the default Mozilla certificate should be used.
- DHCP relay – Required if the server is outside the management interface broadcast domain.

ZTP works in the following network environments:

- nodes, HTTP file servers, and DHCP server in the same subnet
- HTTP file servers and DHCP server in the same subnet, separate from the nodes
- nodes, HTTP file servers, and DHCP server in different subnets

Figure 1: All components in the same subnet shows the first scenario where all components are in a Layer 2 broadcast domain. There is no DHCP relay and all IPs are assigned from a single pool.

Figure 1: All components in the same subnet

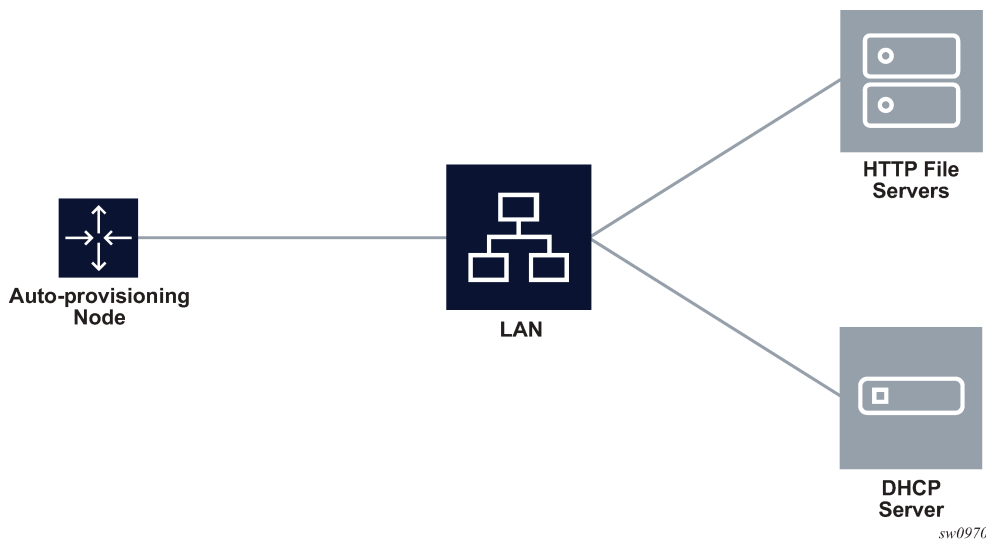


Figure 2: HTTP file and DHCP servers in the same subnet shows the second scenario where only the HTTP file servers and DHCP server are in the same subnet. The DHCP relay is used to fill Option 82 as the gateway address. The gateway address is used to find the appropriate pool in the DHCP server to assign the correct subnet IP address to the SR Linux.

The DHCP offer allows the Option 3 router to define the default gateway. If multiple addresses are provided via Option 3, the first address is used for the default gateway.

Figure 2: HTTP file and DHCP servers in the same subnet

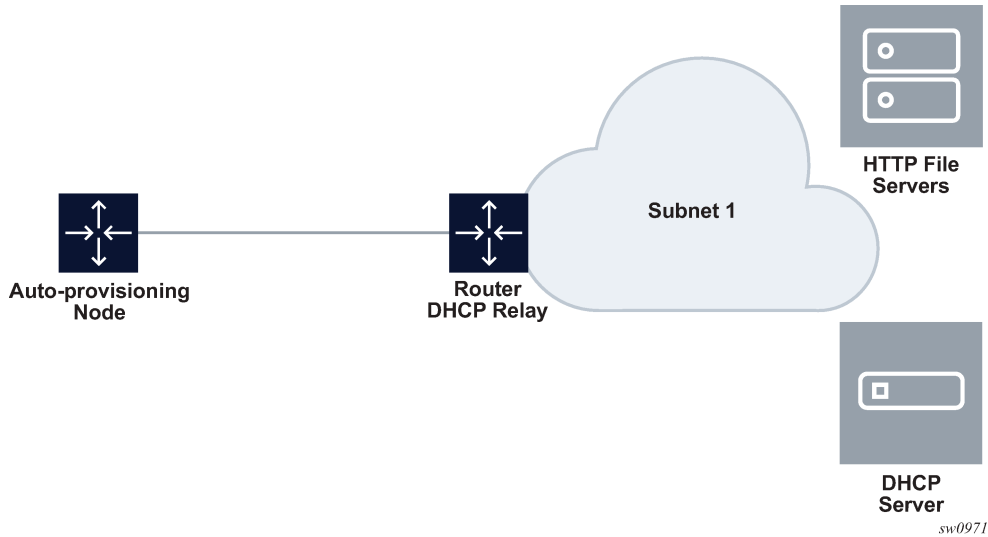
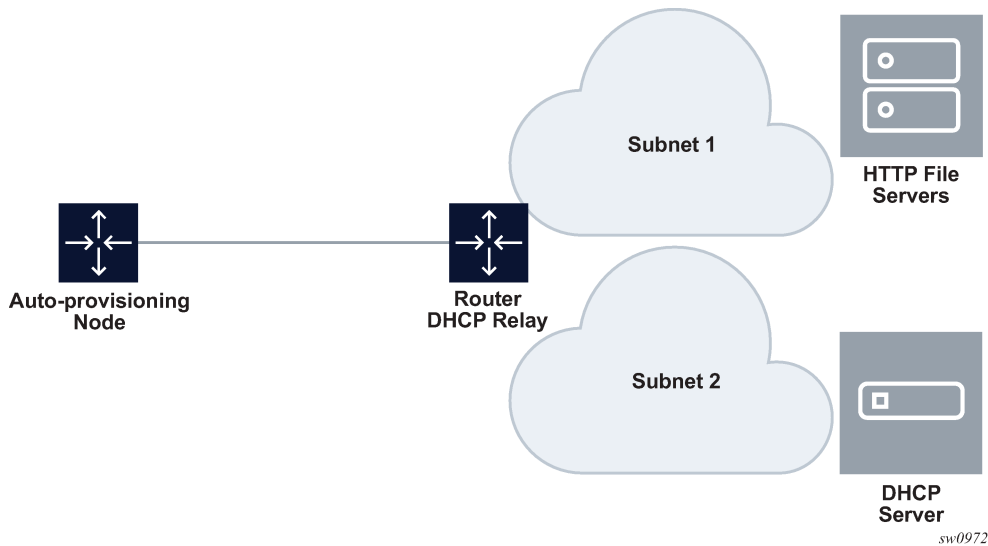


Figure 3: All components in different subnets shows the third scenario where all components are in different subnets. The DHCP relay adds the Option 82 gateway address to the DHCP request, and the DHCP server adds the Option 3 with the gateway address of the HTTP file server.

Figure 3: All components in different subnets



6.3 Process information

When the node reboots, the SR Linux starts the auto-boot process if the auto-boot flag is set in the grub.cfg. Currently only the management port is supported for auto-boot.

6.3.1 DHCP discovery and solicitation

DHCP discovery is sent out from a management port that is operationally up with un-tag format when OOB is used. IPv4 DHCP discovery is sent out first, and if no offer is received within the DHCP timeout, an IPv6 DHCP solicitation is then sent out. The DHCP timeout is 60 seconds.

- For DHCP IPv4, Option 61 is used for pool selection. By default, the node sends Option 61 with the serial number of the chassis.



Note: The grub.cfg can be provisioned with a MAC option as well. When a MAC option is specified, Option 61 is populated with the chassis MAC address.

- For DHCP IPv6, Option 1 is used for pool selection. By default, the node uses RFC 3315 DUID Type 2 vendor-assigned unique ID. The value for *enterprise-id* is 6527 and the identifier is the chassis serial number.



Note: Type 3 is configurable in the grub.cfg.

When the DHCP server receives the discovery packet, it assigns the IP address to the node. The DHCP offer for IPv4 requires the options shown in [Table 3: Required DHCP offer options](#).

Table 3: Required DHCP offer options

| Option | Name | Description |
|--------|----------------------|---|
| viaddr | Client-Ip-Address | Network interface – IP address (for network consistency, a fixed IP address is recommended vs randomly assigned from the DHCP server IP pool) |
| 1 | Subnet Mask | Network interface – Subnet mask |
| 3 | Router | Network interface – Default gateway (Only the first router is used. Additional routers are ignored.) |
| 51 | Lease Time | Validated to be infinite |
| 54 | Server Address | DHCP server identifier |
| 66 | Boot server hostname | Server IP address |
| 67 | Bootfile Name | URL or IP to the provisioning file |

[Table 4: DHCP IPv4 and IPv6 equivalents](#) lists DHCP IPv4 and IPv6 equivalents.

Table 4: DHCP IPv4 and IPv6 equivalents

| Option | IPv4 option | IPv6 Option | IPv6 Comments |
|------------|-------------|--------------------|---|
| Client ID | Option 61 | Option 1 (DUID) | 2 – Vendor-assigned unique ID (default) 3 – Link-layer address |
| NTP server | Option 42 | Option 56 | — |

| Option | IPv4 option | IPv6 Option | IPv6 Comments |
|-------------------------|-------------|-------------|---------------|
| User class | Option 77 | Option 15 | — |
| TFTP server name | Option 66 | NA | — |
| Bootfile name | Option 67 | Option 59 | — |
| Vendor-specific options | Option 43 | Option 17 | — |

6.3.1.1 Auto-provisioning options

Defined options determine how DHCP discovery functions.

The client ID used in IPv4 and IPv6 can be configured as a chassis serial ID or chassis MAC address. By default, the chassis serial ID is used, but the user can configure the auto-boot option to use the chassis MAC address. This option can be configured by editing the grub.cfg and adding the -mac sub-option to the auto-boot option:

grub.cfg location: /nokiaboot/boot/grub2/grub.cfg or /mnt/boot/boot/grub2/grub.cfg

The ZTP timeout default is set at 1 hour for each attempt. During the provisioning process, the node performs three attempts for a period of 3 hours (1 hour for each attempt). After the three initial attempts, the node reboots. The user can change the 1 hour default using the ZTP CLI.

See sections [Configuring ZTP](#) and [ZTP CLI and SR Linux CLI command structures](#) for procedures and commands used to provision available options.

6.3.1.2 DHCP server Option 42 (IPv4) and 56 (IPv6) for NTP

DHCP provides an NTP server URL using Option 42 (for IPv4) or Option 56 (for IPv6). When the time is obtained and synchronized from the NTP server, any log event obtained after this time has the correct timestamp. Any log event before the time was obtained does not have the correct timestamp, and has the default timestamp instead.

6.3.2 DHCP offer

OOB auto-boot supports both IPv4 discovery and IPv6 solicitation, but when the offer is received, the provisioning script follows the same address family format for file download as the DHCP offer.

The first DHCP offer received with the correct Option 66 and 67 or Option 43 is used. The Python provisioning script is downloaded from the location defined by Option 66 and 67 or Option 43.

With an IPv4 DHCP offer, the node IP address and other information, such as the default route, is included.

For IPv6, only the IP arrives from the DHCP server. This offer does not include a prefix, and when it is received, the route is installed with a prefix of /128. This means that the prefix received from the RA is ignored, and the node always acts as a host with a prefix of /128. The default route is received from the Route Advertisement (RA) from the IPv6 peer.

6.3.2.1 Default gateway route configuration for IPv4

After the DHCP offer is received, the Option 3 router can be used to program the default gateway on the SR Linux. A static route should be configured with the default route 0.0.0.0/0 as the next-hop IP address (provided by the Option 3 router).

6.3.2.2 DHCP relay

The DHCP relay can be used to fill Option 82 for the gateway address. The gateway address can be used to find the appropriate pool in the DHCP server to assign the correct subnet IP address to the SR Linux.

6.3.3 Python provisioning script

The Python provisioning file is downloaded from the location dictated by Option 66 and 67 or 43 using HTTP, HTTPS, TFTP, or FTP. Option 66 and 67 takes precedence over Option 43 when both are present in the offer, but Option 43 is used if there is a download error with Option 66 and 67. In addition, Option 66 and 67 can be summarized in Option 67 only. The URL of the provisioning file can be resolved via the DHCP-provided DNS. Up to three DNS servers can be offered by the DHCP.

The node downloads the Python script to storage device. The node then uses the Python provisioning script to download any RPMs, images, scripts, or configurations to the destination dictated by the script.

The URLs defined in the Python script define multiple levels of redundancy. If the primary location is unreachable and times out, two additional redundant servers can be configured. The node cycles through the primary, secondary, and tertiary locations and, when successful, downloads the files to the storage device where they are executed locally.

After successful completion, the provisioning script disables the auto-boot flag to ensure that additional reboots of the node do not enter auto-boot mode. When the nodes reboot, they come up in an operational state with the configuration and image.

For more information about the Python provisioning script, see [Configuring the Python provisioning script](#).

6.3.4 Auto-provisioning failures

The following are possible failure scenarios:

- No Option 66 and 67 or 43 is received (possibly because the format is a URL or no IP address was provided via the DHCP server).
- The download of the Python provisioning script failed or the server was not reachable.
- The download of the RPMs, scripts, or configurations failed (possibly because to the server is not available, or incorrect directory or credentials).
- Copying the RPMs, scripts, or configuration to the storage device failed.

If a failure occurs:

- Details of the failure display on the console and are recorded in the appropriate log files. Log files are stored in the storage device. There can be three log files, which are overwritten in a circular manner.
- The DHCP task is notified of the failure and releases the IP address on the port.

- The auto-boot task goes through the process cycle again until it succeeds, the timeout value is reached, or the auto-boot Option is removed from the grub.cfg, either by editing the grub.cfg or using the ZTP CLI.

6.3.5 ZTP log files

ZTP log files are stored under `/var/log/ztp`.

Example:

```
[root@srlinux ztp]# ls -ltr
total 28528
-rw-r--r-- 1 root root 18220 Sep  4 23:04 ztp_2019-09-04_23-03-52_880867.log
-rw-r--r-- 1 root root 1789 Sep  4 23:29 ztp_2019-09-04_23-29-38_496995.log
-rw-r--r-- 1 root root 18220 Sep  4 23:31 ztp_2019-09-04_23-31-08_996284.log
-rw-r--r-- 1 root root 1791 Sep  5 17:42 ztp_2019-09-05_17-42-01_082002.log
-rw-r--r-- 1 root root 0 Sep  5 21:56 ztp_2019-09-05_21-56-13_730783.log
-rw-r--r-- 1 root root 0 Sep  5 21:56 ztp_2019-09-05_21-56-14_036070.log
[root@srlinux ztp]#
```

6.4 Configuring ZTP

The following are common ZTP configuration procedures:

- [Configuring the Python provisioning script](#)
- [Configuring the ZTP timeout value using the provisioning script](#)

The following are common ZTP configuration procedures using the ZTP CLI:

- [Configuring options in the grub.cfg using ZTP CLI](#)
- [Managing images using ZTP CLI](#)
- [Configuring the NOS using ZTP CLI](#)
- [Redownloading the executable files with ZTP CLI](#)
- [Starting, stopping, and restarting a ZTP process using ZTP CLI](#)
- [Checking the status of a ZTP process using ZTP CLI](#)

The following are common ZTP configuration procedures using the SR Linux CLI:

- [Configuring options in the grub.cfg using SR Linux CLI](#)
- [Specifying the image, kernel, or RAM to boot the system using SR Linux CLI](#)
- [Starting, stopping, and restarting a ZTP process using SR Linux CLI](#)
- [Checking the status of a ZTP process using SR Linux CLI](#)

6.4.1 ZTP CLI versus SR Linux CLI

There are two CLIs where ZTP-related commands can be executed:

- SR Linux CLI (`sr_cli`)

- ZTP CLI

The SR Linux commands are available to use only when the SR Linux is operational at the `sr_cli`.

When the SR Linux is not operational, the ZTP CLI is a unified tool that can be used to manage ZTP tasks at the console.

See the [ZTP CLI and SR Linux CLI command structures](#) sections for a complete list of available ZTP-related commands.

6.4.2 Configuring the Python provisioning script

The primary components of the Python provisioning script include:

- location of provider certificate and trust anchor when HTTPS is used to download RPMs and other bash/Python scripts
- the URL location for each RPM, script, and configuration
- DNS information for resolving the URL of a file. At least one DNS entry is needed for resolving the URL of downloaded files. This DNS can be different from the DHCP offered DNS. If a DNS is defined in the provisioning file, it takes precedence over the DHCP DNS. If there is no DNS in the provisioning script, the DHCP DNS will be used.
- a section to clear the auto-boot option from the `grub.cfg` kernel section

Example: Python provisioning script

```
import errno
import os
import sys
import signal
import subprocess
from subprocess import Popen, PIPE
import threading
srlinux_image_url = 'http://135.227.249.116/srlinux/srlinux-20.6.1-10656.squashfs'
srlinux_image_md5_url = 'http://135.227.249.116/srlinux/srlinux-20.6.1-10656.md5'
srlinux_config_url = 'http://135.227.249.116/srlinux/config.json'
class ProcessError(Exception):
    def __init__(self, msg, errno=-1):
        Exception.__init__(self, msg)
        self.errno = errno
class ProcessOpen(Popen):
    def __init__(self, cmd, cwd=None, env=None, flags=None, stdin=None,
                 stdout=None, stderr=None, universal_newlines=True,):
        self.__use_killpg = False
        shell = False
        if not isinstance(cmd, (list, tuple)):
            shell = True
        # Set flags to 0, subprocess raises an exception otherwise.
        flags = 0
        # Set a preexec function, this will make the sub-process create it's
        # own session and process group - bug 80651, bug 85693.
        preexec_fn = os.setsid
        self.__cmd = cmd
        self.__retval = None
        self.__hasTerminated = threading.Condition()
        Popen.__init__(self, cmd, cwd=cwd, env=env, shell=shell, stdin=stdin,
                       stdout=PIPE, stderr=PIPE, close_fds=True,
                       universal_newlines=universal_newlines, creationflags=flags,)
        print("Process [{}] pid [{}].format(cmd, self.pid))
    def _getReturncode(self):
```

```

    return self.__returncode
def __finalize(self):
    # Any finalize actions
    pass
def _setReturncode(self, value):
    self.__returncode = value
    if value is not None:
        # Notify that the process is done.
        self.__hasTerminated.acquire()
        self.__hasTerminated.notifyAll()
        self.__hasTerminated.release()
returncode = property(fget=_getReturncode, fset=_setReturncode)
def _getRetval(self):
    # Ensure the returncode is set by subprocess if the process is finished.
    self.poll()
    return self.returncode
retval = property(fget=_getRetval)
def wait_for(self, timeout=None):
    if timeout is None or timeout < 0:
        # Use the parent call.
        try:
            out, err = self.communicate()
            self.__finalize()
            return self.returncode, out, err
        except OSError as ex:
            # If the process has already ended, that is fine. This is
            # possible when wait is called from a different thread.
            if ex.errno != 10: # No child process
                raise
            return self.returncode, "", ""
    try:
        out, err = self.communicate(timeout=timeout)
        self.__finalize()
        return self.returncode, out, err
    except subprocess.TimeoutExpired:
        self.__finalize()
        raise ProcessError(
            "Process timeout: waited %d seconds, "
            "process not yet finished." % (timeout)
        )
def kill(self, exitCode=-1, sig=None):
    if sig is None:
        sig = signal.SIGKILL
    try:
        if self.__use_killpg:
            os.killpg(self.pid, sig)
        else:
            os.kill(self.pid, sig)
    except OSError as ex:
        self.__finalize()
        if ex.errno != 3:
            # Ignore:  OSError: [Errno 3] No such process
            raise
    self.returncode = exitCode
    self.__finalize()
def commandline(self):
    """returns string of command line"""
    if isinstance(self.__cmd, six.string):
        return self.__cmd
    return subprocess.list2cmdline(self.__cmd)
__str__ = commandline
def execute_and_out(command, timeout=None):
    print("Executing command: {}".format(command))
    process = ProcessOpen(command)

```

```

try:
    #logger.trace("Timeout = {}".format(timeout))
    ret, out, err = process.wait_for(timeout=timeout)
    return ret, out, err
except ProcessError:
    print("{} command timeout".format(command))
    process.kill()
    return errno.ETIMEDOUT, "", ""
def execute(command, timeout=None):
    ret, _, _ = execute_and_out(command, timeout=timeout)
    return ret
def pre_tasks():
    pass
def srlinux():
    nos_install()
    nos_configure()
def post_tasks():
    pass
def nos_install():
    cmd = 'ztp image upgrade --imageurl {} --md5url {}'.format(srlinux_image_url, srlinux_image_md5_url)
    ret,out,err = execute_and_out(cmd)
def nos_configure():
    cmd = 'ztp configure-nos --configurl {}'.format(srlinux_config_url)
    ret,out,err = execute_and_out(cmd)
def main():
    pre_tasks()
    srlinux()
    post_tasks()
if __name__ == '__main__':
    main()

```

6.4.3 Configuring the ZTP timeout value using the provisioning script

The ZTP process sends DHCP discovery messages on all ports within a ZTP cycle. Every time the DHCP discovery timeout expires and a DHCP offer has not been received, the DHCP discovery process reinitiates on the port until the ZTP timeout expires.

The timeout value can be set using the:

- ZTP CLI (see procedure [Configuring options in the grub.cfg using ZTP CLI](#))
- SR Linux CLI (see procedure [Configuring options in the grub.cfg using SR Linux CLI](#))

6.4.4 Configuring options in the grub.cfg using ZTP CLI

Several options can be manually configured in the grub.cfg using the ZTP CLI at the console. The command has the following format:

```
# ztp option <command> [<arguments>]
```

where *command* must be one of the following:

Table 5: Configuring options

| Command | Description |
|----------|--|
| autoboot | Enables or disables the auto-boot flag |

| Command | Description |
|---------------------|--|
| bootintf | Specifies boot interface options |
| clientid | Sets the client ID to a chassis MAC address or serial ID |
| downgrade | Indicates whether NOS downgrade is allowed |
| duration | Specifies the ZTP timeout value and number of retry attempts |
| formatovl | Indicates the format overlay file system on the next reboot |
| formatsrletc | Indicates the format /etc/opt/srlinux overlay file system |
| formatsrlopt | Indicates the format /opt/srlinux overlay file system |
| list | Displays the current value for each of the command options |
| nosinstall | Specifies whether a NOS upgrade should be performed as part of the ZTP process |
| reload | Reloads the config and updates the grub from the config |
| srlflags | Sets the debug flag in cmdline to a specified value |

[Table 6: ZTP CLI: option command examples](#) describes examples of **ztp option** commands and available arguments.

Table 6: ZTP CLI: option command examples

| Command and description | Command syntax |
|---|--|
| autoboot Enable or disable the auto-boot flag | <code>ztp option autoboot --status [enable disable]</code> |
| bootintf Specify the boot interface to send DHCP over | <code>ztp option bootintf --intf <name of interface></code> |
| bootintf Remove the previously set boot interface | <code>ztp option bootintf --remove</code> |
| duration Set the ZTP timeout value | <code>ztp option duration --timeout <integer in seconds></code> Default = 3600, range = 200 to 3600 |
| duration Set the number of ZTP retry attempts | <code>ztp option duration --retry <integer></code> Default = 3, range = 1 to 10 |
| clientid | <code>ztp option clientid --type serialid</code> |

| Command and description | Command syntax |
|---|--|
| Set the client ID to a serialID | |
| nosinstall Enable or disable NOS upgrade flag | <code>ztp option nosinstall --status [enable disable]</code> |
| list Display the current value of each option | <code>ztp option list</code> |

Example output

The following is an example output of the **ztp option list** command:

```
# ztp option list
+-----+-----+
| Name      | Value  |
+-----+-----+
| autoboot  | False  |
| nosinstall| False  |
| bootintf  | mgmt0  |
| timeout   | 3600   |
| retry     | 3      |
| clientid  | serialid|
| downgrade | True   |
| formatovl | False  |
| formatsrlopt| True  |
| formatsrletc| False |
| srlflags  | None   |
+-----+-----+
```

6.4.5 Managing images using ZTP CLI

The ZTP CLI **ztp image** command can be used to activate, delete, list, or perform an upgrade at the console. The following command format is used:

```
# ztp image <command> [<arguments>]
```




where *command* must be one of the following:

Table 7: Image command descriptions

| Command | Description |
|------------------|--|
| activate | Activate a specific image |
| bootorder | Configure a grub entry to match the boot order as passed |
| delete | Delete a specific image |
| list | List all available NOS images |
| upgrade | Perform an upgrade based on provided parameters |
| version | Extract the version from a specific filename |

[Table 8: ZTP CLI: ztp image command examples](#) describes examples of **ztp image** commands and available arguments.

Table 8: ZTP CLI: ztp image command examples

| Command and description | Command syntax |
|--|---|
| activate Activate a specific NOS image | <pre>ztp image activate --version <build version> [-no-reboot]</pre>  Note: --no-reboot means do not reboot after activate to take new build into use. |
| bootorder Configure the bootorder | <pre>ztp image bootorder --version <build version 1> --version <build version 2> --version <build version 3></pre>  Note: --version means the image version order that booting is attempted, which allows up to 3 versions. |
| delete Delete a specific NOS image | <pre>ztp image delete --version <build version></pre>  WARNING: Active image must not be deleted. |
| list List all available NOS images | <pre>ztp image list</pre> |
| upgrade Download an image for NOS upgrade | <pre>ztp image upgrade --imageurl <URL to download image></pre> |
| upgrade Download an md5 file for NOS upgrade | <pre>ztp image upgrade --md5url <URL to download md5 file></pre> |
| upgrade Do not reboot after an upgrade install | <pre>ztp image upgrade --no-reboot</pre> |
| version Extract a version | <pre>ztp image version --filename <filename path> [-format <format type>]</pre> Example commands: <pre>ztp image version --filename ./config --format json</pre> <pre>ztp image version --filename ./srlinux-20.6.1-12617.tar</pre> |

Example outputs

Example output (list images):

```
[root@localhost ~]# ztp image list
[ 1638.035200] EXT4-
fs (sdb2): mounted filesystem with ordered data mode. Opts: (null)
+-----+
| Versions |
+-----+
| 20.6.1-10654* |
| 20.6.1-10587 |
| 20.6.1 |
+-----+
```

Example output (activate image):

```
[root@localhost ~]# ztp image list
[ 227.172007] EXT4-
fs (sdb2): mounted filesystem with ordered data mode. Opts: (null)
+-----+
| Versions |
+-----+
| 20.6.1-10587* |
| 20.6.1-10654 |
| 20.6.1 |
+-----+
```

Example output (version):

```
[root@localhost ~]# ztp image version --filename ./srlinux-20.6.1-12617.tar
+-----+-----+
| version | message |
+-----+-----+
| 20.6.1-12617 | None |
+-----+-----+
```

6.4.6 Configuring the NOS using ZTP CLI

The ZTP CLI **image** command can be used to push the configuration from the console when SR Linux is not operational.

The following is an example showing how to configure the SR Linux with a configuration downloaded with a user-provided URL:

```
ztp configure-nos --configurl <URL to download configuration>
```

6.4.7 Redownloading the executable files with ZTP CLI

Executable files (RPMs, scripts, and configurations) can be redownloaded if required. The redownload can be performed at the console using the ZTP CLI.

The following is an example showing how to run the provisioning script with a user-provided URL:

```
ztp provision --url <URL where files should be downloaded from>
```

6.4.8 Starting, stopping, and restarting a ZTP process using ZTP CLI

The ZTP process can be manually started, stopped, and restarted using a ZTP CLI command at the console. The following command format is used:

```
# ztp service <command> [<arguments>]
```

where *command* must be one of the following:

Table 9: Starting, stopping, and restarting command descriptions

| Command | Description |
|----------------|---|
| start | Starts ZTP process if not currently running |
| stop | Stops ZTP process if already running |
| restart | Stops ZTP process (if running), and then restarts the process |

Table 10: ZTP CLI: [ztp service command examples](#) describes examples of **ztp service** commands and available arguments.

Table 10: ZTP CLI: ztp service command examples

| Command and description | Command syntax |
|--|---|
| start Start the ZTP process | ztp service start |
| start Start the ZTP process and enable/disable auto-boot | ztp service start --autoboot [<i>enable</i> <i>disable</i>] |
| stop Stop the ZTP process | ztp service stop |
| stop Stop the ZTP process and disable auto-boot | ztp service stop --autoboot disable |
| restart Restart the ZTP process | ztp service restart |
| restart Restart the ZTP process and enable/disable auto-boot | ztp service restart --autoboot [<i>enable</i> <i>disable</i>] |

6.4.9 Checking the status of a ZTP process using ZTP CLI

The ZTP process status can be manually checked using the ZTP CLI command at the console.

The following is an example showing how to check the status of the ZTP process:

ztp service status

Example:

```
# ztp service status
+-----+-----+
| Service | Status |
+-----+-----+
| ztp     | Active |
+-----+-----+
#
```

6.4.10 Configuring options in the grub.cfg using SR Linux CLI

Several auto-boot related options can be manually configured in the grub.cfg using the SR Linux CLI when SR Linux is operational. The command has the following format:

```
# system boot autoboot <command> [<arguments>]
```

where *command* must be one of the following:

Table 11: Configuring options

| Command | Description |
|--------------------|--|
| admin-state | Enables or disables the auto-boot functionality |
| interface | Sets the interface used for the auto-boot functionality |
| timeout | Sets the timeout for each auto-boot attempt |
| attempts | Sets the amount of auto-boot executions to try before rebooting the system |
| client-id | Sets the client ID to use on outgoing DHCP requests |

Table 12: [SR Linux CLI: autoboot commands for grub.cfg update examples](#) describes examples of SR Linux **autoboot** commands and available arguments.

Table 12: SR Linux CLI: autoboot commands for grub.cfg update examples

| Command and description | Command syntax |
|--|---|
| admin-state Enable or disable the auto-boot flag | system boot autoboot admin-state [<i>enable</i> <i>disable</i>] Default = enable |

| Command and description | Command syntax |
|--|---|
| interface Specify the boot interface to send DHCP over | system boot autoboot interface <i><name of interface></i> Default = mgmt0 |
| timeout Set the ZTP timeout value | system boot autoboot timeout <i><integer in seconds></i> Default = 3600, range = 200 to 3600 |
| attempts Set the number of ZTP retry attempts | system boot autoboot attempts <i><integer></i> Default = 3, range = 1 to 10 |

6.4.11 Specifying the image, kernel, or RAM to boot the system using SR Linux CLI

Users can specify an ordered list of local images, kernels, or initial RAM disks to boot the system using the SR Linux CLI when SR Linux is operational. This directly translates into boot configuration in the grub, where the images or kernels are tried in the order specified by the user. The command has the following format:

```
# tools system boot <command> [<arguments>]
```

where *command* must be the following:

Table 13: Specifying the image, kernel, or RAM

| Command | Description |
|--------------|---|
| image | User-specified ordered list of local images used to boot the system |

[Table 14: SR Linux CLI: image and kernel boot command example](#) describes an example of the SR Linux **image** and **kernel boot** command and available argument.

Table 14: SR Linux CLI: image and kernel boot command example

| Command and description | Command syntax |
|--|---|
| image Specify an ordered list of images to boot the system | tools system boot image <i><ordered list of images></i> Note: Up to 3 files can be specified. |

6.4.12 Starting, stopping, and restarting a ZTP process using SR Linux CLI

The ZTP process can be manually started, stopped, and restarted using the SR Linux CLI when SR Linux is operational. The following command format is used:

```
# tools system boot autoboot <command>
```

where *command* must be one of the following:

Table 15: Commands

| Command | Description |
|-----------------------|---|
| execute-script | Executes a specified script as if it were received during auto-boot |
| start | Starts a ZTP process if not currently running |
| stop | Stops a ZTP process if already running |
| restart | Stops an in-progress auto-boot process, then initiates another |

Table 16: SR Linux CLI: start, stop, and restart process command examples describes examples of SR Linux **start**, **stop**, and **restart** commands and available arguments.

Table 16: SR Linux CLI: start, stop, and restart process command examples

| Command and description | Command syntax |
|---|--|
| execute-script Execute a specified script | <code>tools system boot autoboot execute-script <URL to the script></code> |
| start Start the ZTP process | <code>tools system boot autoboot start</code> |
| stop Stop the ZTP process | <code>tools system boot autoboot stop</code> |
| restart Restart the ZTP process | <code>tools system boot autoboot restart</code> |

6.4.13 Checking the status of a ZTP process using SR Linux CLI

The ZTP process status can be manually checked using the SR Linux CLI when SR Linux is operational.

The following is an example showing how to check the status of the ZTP process:

```
# tools system boot autoboot status
```

6.5 ZTP CLI and SR Linux CLI command structures

This section describes the ZTP CLI command structure and SR Linux CLI command structure.

6.5.1 ZTP CLI command structure

The following ZTP CLI commands are available at the console:

```
ztp
  - chassis
    - control
      - [--format table | json]
    - linecards
      - [--format table | json]
  configure-nos
    - --configurl <URL to download configuration>
  image
    - activate
      - --version <build version>
      - [--no-reboot]
    - bootorder
      - --version <up to 3 build versions>
    - delete
      - --version <build version>
    - list
      - [--format table | json]
    - upgrade
      - --imageurl <URL to download image> --md5url <URL to download md5 file>
      - [--no-reboot]
      - [--skip-check]
      - [--not-active]
    - version
      - --filename <name>
      - [--format table | json]
  option
    - autoboot
      - --status enable | disable
    - bootintf
      - --intf <boot interface>
    - clientid
      - --type serialid
    - downgrade
      - --status enable | disable
    - duration
      - --timeout <seconds> --retry <integer>
    - formatall
      - --status enable | disable
    - formatovl
      - --status enable | disable
    - formatsrletc
      - --status enable | disable
    - formatsrlopt
      - --status enable | disable
    - grubopt
      - [--key <text>]
      - [--value <text>]
      - [--delete]
    - list
      - [--format table | json]
```

```

- nosinstall
  - --status enable | disable
- reload
- srlflags
  - [--value <text>]
  - [--delete]
provision
  - --url <URL to download provisioning script>
service
  - restart
    - --autoboot enable | disable
  - start
    - --autoboot enable | disable
  - status
    - [--format table | json]
  - stop
    - --autoboot enable | disable

```

6.5.2 SR Linux CLI command structure

The following SR Linux auto-boot related tool commands are available when the SR Linux is operational:

```

system
- boot
  - autoboot
    - admin-state
    - attempts
    - client-id
    - interface
    - status
    - timeout
  - image

```

```

tools
- system
  - boot
    - autoboot
      - execute-script
      - restart
      - start
      - status
      - stop

```

Refer to the *SR Linux Data Model Reference* for additional information about SR Linux commands and parameter descriptions.

Appendix: ZTP Python library

This appendix describes the importable ZTP Python library.

For more information about the ZTP process, see [Zero Touch Provisioning](#).

ZTPClient

The ZTPClient communicates with the SR Linux ZTP process. The APIClient is the core object of the ZTPClient. Each use of the ZTPClient passes through a call to one of its methods.

The path to the API client class is as follows:

```
class ztpclient.ztpclient.APIClient(base_url=None)
```

Example

```
def __init__(self):
    self.client = ztpclient.APIClient()
def get_option(self, item):
    ret = self.client.option_list()
    return ret['message'].get(item, None)
def find_current_version(self):
    response = self.client.image_list()
    if response:
        image = response['message']
        if image and isinstance(image, list) and len(list) > 0:
            return image[0].replace('*', '')
    return None
def perform_ztp(self):
    self.nos_install()
    self.nos_configure()
    self.disable_autoboot()
def nos_install(self):
    ret = self.client.image_upgrade(srlinux_image_url, srlinux_image_md5_url)
    if ret:
        return int(ret['status'])
    return -1
def nos_configure(self):
    ret = self.client.configure(srlinux_config_url)
    if ret:
        return int(ret['status'])
    return -1
def disable_autoboot(self):
    ret = self.client.option_autoboot(ztpclient.ZtpStatus.disable)
    if ret:
        return int(ret['status'])
    return -1
if __name__ == '__main__':
    ztp = ZTP()
    ztp.perform_ztp()
```

Functions

This section describes the possible functions.

chassis_control()

Lists control card information.

Table 17: *chassis_control()*

| Information | Description |
|-------------|---|
| Arguments | — |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. The message attribute contains dictionary with control card information. |
| Example | <pre>>>> client.chassis_control() {'status': 0, u'message': {'operation': u'active'}} >>> client.chassis_control() {'status': 0, u'message': {'operation': u'standby'}}</pre> |

chassis_linecards()

Lists line card information for the chassis.

Table 18: *chassis_linecards()*

| Information | Description |
|-------------|--|
| Arguments | — |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. The message attribute contains list of dicts, where list item is dict with line card information. |
| Example | <pre>>>> client.chassis_linecards() {'status': 0, u'message': [{u'card_type': 127, u'card_name': u'imm32-100g-qsfp28+4-400g-qsfpdd', u'slot_num': 1}, {u'card_type': 0, u'card_name': u'empty', u'slot_num': 2}, {u'card_type': 0, u'card_name': u'empty', u'slot_num': 3}, {u'card_type': 127, u'card_name': u'imm32-100g-qsfp28+4-400g-qsfpdd', u'slot_num': 4}]}</pre> |

configure(configurl)

Downloads the configuration from a specific **configurl** and applies the configuration to SR Linux. If SR Linux services are not running, the services are started and the configuration is applied.


Table 19: *configure(configurl)*

| Information | Description |
|-------------|---|
| Arguments | configurl (string): the URL from where the configuration will be downloaded |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Example | — |

image_activate(version)

Reboots the chassis to the image version provided. If the current active version is the same as the specified **version**, no action is performed. If there is no image in the chassis of the specified **version**, no action is performed. If the specified **version** is available, the chassis will be rebooted to that **version**.

Table 20: *image_activate(version)*

| Information | Description |
|-------------|--|
| Arguments | version (string): the image version |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise.  Note: This API may result in a chassis reboot to activate the image version. |
| Examples | <pre>>>> client.image_list() {'status': 0, 'message': [u'20.6.1-18836*', u'20.6.1-17740', u'20.6.1-17738']} >>> client.image_activate('20.6.1-3333') {'status': 127, 'message': u'20.6.1-3333 is not available'} >>> client.image_activate('20.6.1-18836') {'status': 127, 'message': u'20.6.1-18836 is current active version. No additional change required'}</pre> |

image_bootorder(bootorder)

Sets the image bootorder in the Grub configuration. On the next reboot, the chassis reboots to the first image in the list.

Table 21: `image_bootorder(bootorder)`

| Information | Description |
|-------------|--|
| Arguments | bootorder (list): the image version list |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.image_bootorder(['20.6.1-18836', '20.6.1-17740', '20.6.1-17738']) {'status': 0, u'message': None} >>> client.image_bootorder('20.6.1-18836,20.6.1-17740,20.6.1-17738') {'status': 0, u'message': None}</pre> |

`image_delete(version)`

Removes the specified image **version** from the chassis. If the specified **version** is not available in the chassis, no action is performed. If the specified **version** is the current active version in the chassis, no action is performed.

Table 22: `image_delete(version)`


| Information | Description |
|-------------|---|
| Arguments | version (string): the image version |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.image_list() {'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740', u'20.6.1-17738']} >>> client.image_delete('20.6.1-3333') {'status': 0, u'message': u'20.6.1-3333 version not available'} >>> client.image_delete('20.6.1-18836') {'status': 127, u'message': u'Cannot remove active version'}</pre> |

`image_list()`

Lists all currently available image versions on the hardware.

Table 23: `image_list()`



| Information | Description |
|-------------|-------------|
| Arguments | — |

| Information | Description |
|-------------|--|
| Returns | <p>(dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. The message attribute contains the list of images. The list item followed by an asterisk (*) indicates the current active image version.</p> <p> Note: The image_list does not indicate the boot order.</p> |
| Examples | <pre>>>> client.image_list() {'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740', u'20.6.1-17738']}</pre> |

image_upgrade(image_url, md5_url, options)

Performs an image upgrade.

Table 24: image_upgrade(image_url, md5_url, options)

| Information | Description |
|-------------|--|
| Arguments | <p>image_url (string): the URL from where the image should be downloaded</p> <p>md5_url (string): The URL from where the pre calculated md5sum of the image should be downloaded. After the image is downloaded, the calculated md5sum is checked against the downloaded md5sum. If the values do not match, the image upgrade is discarded.</p> <p>no_reboot (boolean): If set to true, a chassis reboot is not triggered after an image upgrade. The new image will not be taken into use until the next reboot. The default is false.</p> <p>skip_check (boolean): If set to true, the status check of the autoboot parameter is skipped, and a forced upgrade is performed. If set to false, the image upgrade will only be performed if autoboot is enabled. The default is false.</p> <p>not_active (boolean): If set to true, after an image install, the image will not be marked as the active image (that is, will not reboot to the upgrade image). The current working image is still marked as active. The default is false.</p> <p> Note: Based on the setting and outcome, the chassis can be rebooted when invoking this API.</p> <p> Note: To perform ZTP, the autoboot flag must be enabled.</p> |
| Returns | <p>(dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise.</p> |
| Examples | — |

option_autoboot(status)

Sets the **autoboot** option status. This option determines if **autoboot** should be performed during ZTP. If disabled, ZTP skips all steps and starts the SR Linux application.

Table 25: option_autoboot(status)

| Information | Description |
|-------------|---|
| Arguments | status (ZtpStatus): ztpclient.ZtpStatus.enable to enable the option, ztpclient.ZtpStatus.disable otherwise |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_autoboot(ztpclient.ZtpStatus.enable) {'status': 0, u'message': None}</pre> |

option_bootintf(interface)

Sets the interface to be used by ZTP in various procedures. The default value is **mgmt0**.

Table 26: option_bootintf(interface)

| Information | Description |
|-------------|---|
| Arguments | interface (string): the Linux network interface name |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_bootintf('mgmt0') {'status': 0, u'message': None}</pre> |

option_clientid(type)

Sets the client ID used by ZTP when performing a DHCP request. The possible values are **serialid** and **mac**. When **serialid** is selected, the chassis serial number is used as the client ID in the DHCP request. When **mac** is selected, the Linux interface hardware address (chassis MAC address) is used as client identifier.

Table 27: option_clientid(type)

| Information | Description |
|-------------|---|
| Arguments | type (ZtpClientId): the client identifier type |

| Information | Description |
|-------------|---|
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_clientid(ztpclient.ZtpClientId.serialid) {'u'status': 0, u'message': None} >>> client.option_clientid(ztpclient.ZtpClientId.mac) {'u'status': 0, u'message': None}</pre> |

option_downgrade(status)

Sets the **downgrade** option status of ZTP. When enabled, the option allows ZTP to perform a downgrade of the image (that is, move from a later version image to an earlier version). When the option is disabled, only upgrades are allowed.

Table 28: *option_downgrade(status)*

| Information | Description |
|-------------|---|
| Arguments | status (ZtpStatus): ztpclient.ZtpStatus.enable to enable the option, ztpclient.ZtpStatus.disable otherwise |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_downgrade(ztpclient.ZtpStatus.enable) {'u'status': 0, u'message': None}</pre> |

option_duration(timeout, retry)

Sets the **timeout** and **retry** parameters of the ZTP process. If not successful, the ZTP process keeps retrying for the specified **timeout** seconds. When the timeout is reached, the process stops. If the number of attempts are equal to the **retry** value, the specified action is taken. The default action is to reboot.

Table 29: *option_duration(timeout, retry)*

| Information | Description |
|-------------|--|
| Arguments | timeout (int): the number of seconds to perform ZTP before it is marked as failed retry (int): the number of attempts before stopping the ZTP process |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_duration(3600,3)</pre> |

| Information | Description |
|-------------|----------------------------------|
| | {u'status': 0, u'message': None} |

option_formatovl(status)

Sets the **formatovl** option status of ZTP. When enabled, the option sets the **srl.formatovl** flag in the Grub configuration. On the next reboot, if the **srl.formatovl** flag is set, the NOKIA-DATA overlay file system is formatted. Any change performed on the overlay file system is removed.

Table 30: option_formatovl(status)

| Information | Description |
|-------------|---|
| Arguments | status (ZtpStatus): ztpclient.ZtpStatus.enable to enable the option, ztpclient.ZtpStatus.disable otherwise |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_formatovl(ztpclient.ZtpStatus.enable) {u'status': 0, u'message': None}</pre> |

option_formatsrletc(status)

Sets the **formatsrletc** option status of ZTP. When enabled, the option sets the **srl.formatetc** flag in the Grub configuration. On the next reboot, if the **srl.formatetc** flag is set, the NOKIA-ETC overlay file system is formatted. Any change performed on the overlay file system will be removed.

Table 31: option_formatsrletc(status)

| Information | Description |
|-------------|---|
| Arguments | status (ZtpStatus): ztpclient.ZtpStatus.enable to enable the option, ztpclient.ZtpStatus.disable otherwise |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_formatsrletc(ztpclient.ZtpStatus.enable) {u'status': 0, u'message': None}</pre> |

option_formatsrlopt(status)

When enabled, the option sets the **srl.formatopt** flag in the Grub configuration. On the next reboot, if the **srl.formatopt** flag is set, the NOKIA-OPT overlay file system is formatted. Any change performed on the overlay file system will be removed.

Table 32: option_formatsrlopt(status)

| Information | Description |
|-------------|---|
| Arguments | status (ZtpStatus): ztpclient.ZtpStatus.enable to enable the option, ztpclient.ZtpStatus.disable otherwise |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_formatsrlopt(ztpclient.ZtpStatus.enable) {u'status': 0, u'message': None}</pre> |

option_list()

Lists all the options of the ZTP process.

Table 33: option_list()

| Information | Description |
|-------------|---|
| Arguments | — |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_list() {u'status': 0, u'message': {u'formatsrletc': False, u'retry': 3, u'bootintf': u'mgmt0', u'clientid': u'serialid', u'autoboot': False, u'srlflags': u'no-reboot', u'formatovl': False, u'formatsrlopt': False, u'timeout': 3600, u'downgrade': True, u'nosinstall': False}}</pre> |

option_nosinstall(status)

Sets the **nosinstall** option status. This option determines if an image upgrade should be performed during ZTP. Only the image upgrade step is skipped. All other steps of ZTP are still performed.



Table 34: `option_nosinstall(status)`

| Information | Description |
|-------------|---|
| Arguments | status (ZtpStatus): <code>ztpclient.ZtpStatus.enable</code> to enable the option, <code>ztpclient.ZtpStatus.disable</code> otherwise |
| Returns | (dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.option_nosinstall(ztpclient.ZtpStatus.enable) {'status': 0, u'message': None}</pre> |

`provision(provisionurl)`

Downloads the provision script from a specific **provisionurl** and executes the script. The script could be either **Python** or **Bash**.

Table 35: `provision(provisionurl)`

| Information | Description |
|-------------|--|
| Arguments | provisionurl (string): the URL from where the provisioning script will be downloaded |
| Returns | <p>(dict) The API response as a Python dictionary. The status attribute is set to 0 if successful, or a non-zero value otherwise.</p> <p> Note: If the script returns a non-zero exit code, the status attribute in the return dictionary is set to non-zero. It could be possible that the provisioning script has a chassis reboot command and a chassis will reboot while executing this API.</p> <p> Note: To perform ZTP, the autoboot flag must be enabled.</p> |
| Examples | <pre>>>> client.provision('http://135.227.248.118/duts/IDNS1833F0766/ srlinux_ztp.py')</pre> |

`service_restart()`

Restarts the ZTP service.

Table 36: `service_restart()`

| Information | Description |
|-------------|-------------|
| Arguments | — |

| Information | Description |
|-------------|---|
| Returns | (dict) The API response as a Python dictionary, including the service status in the message attribute. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.service_restart() {'status': 0, u'message': {u'status': u'Service started'}}</pre> |

service_start()

Starts the ZTP service (if not already running).

Table 37: service_start()

| Information | Description |
|-------------|---|
| Arguments | — |
| Returns | (dict) The API response as a Python dictionary, including the service status in the message attribute. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.service_start() {'status': 0, u'message': {u'status': u'Service started'}}</pre> |

service_status()

Gets the current status of the ZTP service. The ZTP service will be running as a systemd service. It can be checked manually by executing the **systemctl status ztp** command.

Table 38: service_status()

| Information | Description |
|-------------|---|
| Arguments | — |
| Returns | (dict) The API response as a Python dictionary, including the service status in the message attribute. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.service_status() {'status': 0, u'message': {u'status': u'Inactive'}} >>> client.service_status() {'status': 0, u'message': {u'status': u'Active'}}</pre> |

service_stop()

Stops the ZTP service (if already running).

Table 39: service_stop()

| Information | Description |
|-------------|---|
| Arguments | — |
| Returns | (dict) The API response as a Python dictionary, including the service status in the message attribute. The status attribute is set to 0 if successful, or a non-zero value otherwise. |
| Examples | <pre>>>> client.service_stop() {u'status': 0, u'message': {u'status': u'Service stopped'}}</pre> |

Appendix: Migrating from CentOS to Debian OS

The migration process from CentOS to Debian OS involves the following high-level changes:

- The filename format for `.deb` packages differs from that of `.rpm` packages. In the `.deb` format, the filename follows the pattern: `<name>_<version>_<arch>.deb`



Note: Underscores must not be included in *name*, *version*, or *arch* components. The `.rpm` format is different from this, as it uses hyphens to separate components and allows underscores in names.

In addition to change in package name, this also involves change in package formatting, including a change in the package manager.

- Debian uses the **apt** or **apt-get** command for package management, while CentOS uses **yum** or **dnf**.
- The loading of shell profile files, such as `/etc/profile.d/sr_app_env.sh`, differs between Debian and CentOS. In most cases, the loading process is centrally aligned. However, there can be instances where the environment of an application or script does not meet the expected conditions. For example, the **PATH** variable may not include the **srlinux** paths. In such cases, you can replace **sh** to **bash**, or append **-l** to the shell command, or combine both options, to make it a login shell.

Debian 12 uses Python 3.11.2 by default. The Python upgrade from the previously used 3.6 version to 3.11.2 does not affect the functionality of the following items, unless otherwise specified.

- Event Handler scripts
- NDK



Note: The new path of the generated NDK protos on Debian is `/usr/lib/python3.11/dist-packages/sdk_protos`. A symlink has been added to refer to the new path from the old one. In future releases, the symlink will no longer be available. Therefore, NDK developers who use on-box generated protos must update their import paths accordingly.

- Python provisioning scripts
- Show commands and CLI plug-ins
- ZTP Python library

However, in the case of custom package development, following must be considered:

- Python packages are installed as `wheel`s instead of `eggs`. For instructions about packaging in `wheel`s format, see the [Wheels binary package documentation](#).
- The parametrized package replaces the Python module `nose-parameterized` package. For migration instructions, see the [nose-parametrized](#) documentation.

Customer document and product support



Customer documentation

[Customer documentation welcome page](#)



Technical support

[Product support portal](#)



Documentation feedback

[Customer documentation feedback](#)