NOKIA

Nokia Service Router Linux

SOFTWARE INSTALLATION GUIDE
Release 20.6

3HE 16113 AAAA TQZZA

Edition: 2

November 2020

Table of contents

1 Getting started

This chapter provides an overview of this document, includes summaries of changes from previous releases, and lists precautionary messages and command conventions.

1.1 About this document

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.

1.2 Summary of changes

The following changes were made in this release.

Table 1 **Change summary**

Topic	Location
Installation information for the 7220 IXR-D system added	Installing software
Importable ZTP Python library appendix added	Appendix: ZTP Python library

1.3 Precautionary messages

Observe all dangers, warnings, and cautions in this document to avoid injury or equipment damage during installation and maintenance. Follow the safety procedures and guidelines when working with and near electrical equipment.

Table 2 describes information symbols contained in this document.

Table 2 **Information symbols**

Symbol	Meaning	Description
⚡	Danger	Warns that incorrect handling and installation could result in bodily injury. An electric shock hazard could exist. Before beginning work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.
⚠	Warning	Warns that incorrect handling and installation could result in equipment damage or loss of data.
⛔	Caution	Warns that incorrect handling may reduce component or system performance.
➡	Note	Notes contain suggestions or additional operational information.

1.4 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**.
- Angle brackets (< >) indicate an item that is not used verbatim. For example, for the command **show ethernet <***name***>**, *name* should be replaced with the name of the interface.
- 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 Introduction

This chapter provides an overview of basic software operations on the SR Linux.

2.1 File system layout

The file system is distributed and laid out as a closed-source third-party application on the OS. The directory structure is listed in Table 3.

Table 3 **File system layout**

Path	Purpose
/opt/srlinux/bin/*	Application binaries
/opt/srlinux/lib/*	Shared libraries
/etc/opt/srlinux/config.json	System configuration
/etc/opt/srlinux/banner	The system banner, pre-login
/etc/opt/srlinux/srlinux.rc	Global environment file
/etc/opt/srlinux/tls/*	User-configured certificates
/etc/opt/srlinux/appmgr/*	YAML configuration
/etc/opt/srlinux/yang/*	YANG models
/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

Table 3 **File system layout (Continued)**

Path	Purpose
/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.

2.2 Boot process

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 general boot sequence:

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 CentOS 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 will wait 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 Installing containers

This chapter provides an overview of container installation tasks. Container installation tasks include:

- Launching a container manually

  Launches a single SR Linux container using a manual procedure.
- Launching a container topology

  Launches a container topology using an automated script.

3.1 Container installation prerequisites

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

Minimum system requirements:

- Centos 7.4 system, Kernel 4.17
- 8 GB RAM
- 8 core CPU
- The host machine user should have sudo privileges (preferably passwordless)

Minimum software requirements:

- Docker CE installed, minimum version 18.09:

  https://docs.docker.com/get-docker/
- Python version 3.6 or higher
- license key file available
- docker-topo-master.tgz file (if building SR Linux container topology)
- srlinux-v*X*.*Y*.*Z*-*N*.tar.xz file (for the SR Linux docker image)

  where *X*=Major, *Y*=Minor, *Z*=Patch, and *N*=Build Number

3.2 Launching a container manually

This procedure manually launches a single container.

Step 1. Copy the *srlinux-vX.Y.Z-N.tar.xz* file into *~/* of the Centos 7 Host Machine.

Step 2. Copy the license.key into *~/license.key*.

Step 3. Load the docker image. To load the image, the user must have root privilege, or be part of the docker group.

```
$ docker image load -i ~/srlinux-vX.Y.Z-N.tar.xz
```

Step 4. Check that the docker image was imported correctly:

```
$ docker images
```

Example:

```
REPOSITORY TAG          IMAGE ID              CREATED         SIZE
srlinux 19.11.1-        3e77d45745f2          7 hours ago     1.3GB
```

Step 5. Launch an instance of the SR Linux container on the host using the options in the following command line. This command must be entered in a single line. See the **Note** that follows about copying text from a PDF file without broken lines.

```
$ docker run -t -d --rm --privileged \
--sysctl net.ipv6.conf.all.disable_ipv6=0 \
--sysctl net.ipv4.ip_forward=0 \
--sysctl net.ipv6.conf.all.accept_dad=0 \
--sysctl net.ipv6.conf.default.accept_dad=0 \
--sysctl net.ipv6.conf.all.autoconf=0 \
--sysctl net.ipv6.conf.default.autoconf=0 \
-u $(id -u):$(id -g) \
-v ~/license.key:/opt/srlinux/etc/license.key:ro \
--name srlinux_dut1 srlinux:vX.Y.Z-N
sudo bash -c '/opt/srlinux/bin/sr_linux'
```

where *X*=Major, *Y*=Minor, *Z*=Patch, and *N*=Build Number for `srlinux:vX.Y.Z-N`

Note: Copying a long command string from a PDF file will introduce line breaks. As a workaround, copy the text string and place into Notepad++. Highlight the text and select CTRL+J. The result is a single line with no returns.

Step 6. Check that the docker container has been created with the name 'srlinux':

```
$ docker ps
```

Example:

```
CONTAINER ID  IMAGE               COMMAND             CREATED     STATUS   PORTS   NAMES
9d5dbd03f7f8  srlinux:19.11.1     "/tini--fixuid-q.." 3 mins ago  Up3 mins.     srlinux_dut1
```

Step 7. Turn off the Docker0 Tx checksum offload:

```
# sudo ethtool --offload docker0 tx off
```

Step 8. Open an SSH session to the DUT using the following credentials:

- username: `admin`
- password: `admin`

Example:

```
# ssh admin@$(docker inspect srlinux_dut1 --format {{.NetworkSettings.IPAddress}})
admin@172.17.0.4's password:
      ___         ___                               ___         ___         ___
     /  /\       /  /\                  ___        /__/\       /__/\       /__/|
    /  /:/_     /  /::\  ___           /  /\       \  \:\      \  \:\      |  |:|
   /  /:/ /\   /  /:/\:\/__/\         /  /:/        \  \:\      \  \:\     |  |:|
  /  /:/ /::\ /  /:/~/:/\  \:\     __/__/::\    _____\__\:\ ___  \  \:\  __|__|:|
 /__/:/ /:/\:/__/:/ /:/__\  \:\   /  \__\/\:\  /__/::::::::/__/\  \__\:/__/::::\____
 \  \:\/:/~/:\  \:\/:::::/\  \:\ /  /:/ \  \:\/\  \:\~~\~~\\  \:\ /  /:/  ~\~~\::::/
  \  \::/ /:/ \  \::/~~~~  \  \:\  /:/   \  \:\/\  \:\  ~~~ \  \:\  /:/    |~~|:|~~
   \__\/ /:/   \  \:\       \  \:\/:/     \__\::/\  \:\      \  \:\/:/     |  |:|
     /__/:/     \  \:\       \  \::/      /__/:/  \  \:\      \  \::/      |  |:|
     \__\/       \__\/        \__\/       \__\/    \__\/       \__\/       |__|/
Hello admin,
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ running }--[  ]--
```

Step 9. Verify the application versions running on the system:

```
# info from state system app-management application * |
as table | filter fields pid author version
```

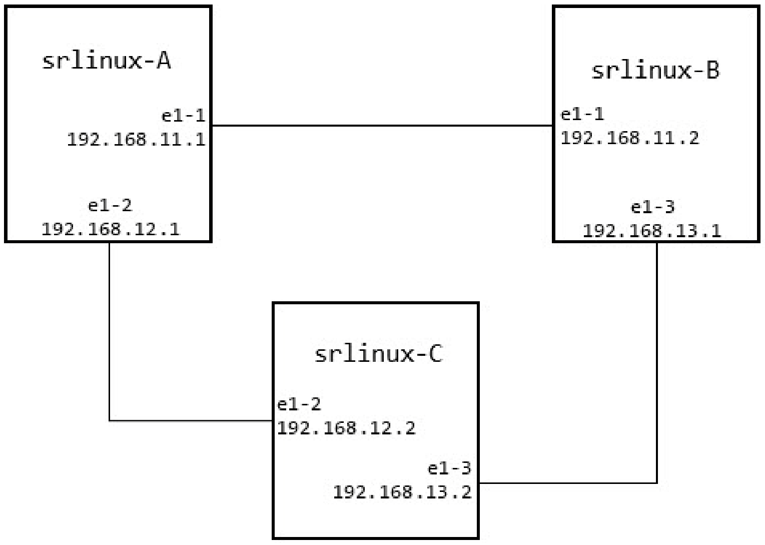
Example:

```
A:3-node-srlinux-A# info from state system app-management application * |
as table | filter fields pid author version
+---------------------+---------+------------+------------------------+
|        Name         |   Pid   |  Author    |        Version         |
+=====================+=========+============+========================+
| aaa_mgr             |    2189 | Nokia      | v20.6.0-13-gc6a313b84c |
| acl_mgr             |    2198 | Nokia      | v20.6.0-13-gc6a313b84c |
| app_mgr             |    2131 | Nokia      | v20.6.0-13-gc6a313b84c |
| arp_nd_mgr          |    2207 | Nokia      | v20.6.0-13-gc6a313b84c |
| bfd_mgr             |         |            |                        |
| bgp_mgr             |    2650 | Nokia      | v20.6.0-13-gc6a313b84c |
| chassis_mgr         |    2216 | Nokia      | v20.6.0-13-gc6a313b84c |
| dev_mgr             |    2155 | Nokia      | v20.6.0-13-gc6a313b84c |
| dhcp_client_mgr     |    2228 | Nokia      | v20.6.0-13-gc6a313b84c |
| fib_mgr             |    2237 | Nokia      | v20.6.0-13-gc6a313b84c |
| gnmi_server         |    2473 | Nokia      | v20.6.0-13-gc6a313b84c |
| idb_server          |    2180 | Nokia      | v20.6.0-13-gc6a313b84c |
| isis_mgr            |         |            |                        |
| json_rpc            |    2476 | Nokia      | v20.6.0-13-gc6a313b84c |
| l2_mac_learn_mgr    |    2248 | Nokia      | v20.6.0-13-gc6a313b84c |
| l2_mac_mgr          |    2258 | Nokia      | v20.6.0-13-gc6a313b84c |
| l2_static_mac_mgr   |         |            |                        |
| lag_mgr             |    2267 | Nokia      | v20.6.0-13-gc6a313b84c |
| linux_mgr           |    2276 | Nokia      | v20.6.0-13-gc6a313b84c |
| lldp_mgr            |    2657 | Nokia      | v20.6.0-13-gc6a313b84c |
| log_mgr             |    2285 | Nokia      | v20.6.0-13-gc6a313b84c |
| mcid_mgr            |    2294 |            |                        |
| mgmt_server         |    2303 | Nokia      | v20.6.0-13-gc6a313b84c |
| mpls_mgr            |         |            |                        |
| net_inst_mgr        |    2312 | Nokia      | v20.6.0-13-gc6a313b84c |
| oam_mgr             |    2325 | Nokia      | v20.6.0-13-gc6a313b84c |
| ospf_mgr            |         | Nokia      |                        |
| plcy_mgr            |    2667 | Nokia      | v20.6.0-13-gc6a313b84c |
| qos_mgr             |    2471 | Nokia      | v20.6.0-13-gc6a313b84c |
| sdk_mgr             |    2335 | Nokia      | v20.6.0-13-gc6a313b84c |
| sshd-mgmt           |    2923 |            |                        |
| static_route_mgr    |    2674 | Nokia      | v20.6.0-13-gc6a313b84c |
| supportd            |    2140 | Nokia      | v20.6.0-13-gc6a313b84c |
| vrrp_mgr            |         |            |                        |
| xdp_cpm             |    2349 | Nokia      | v20.6.0-13-gc6a313b84c |
| xdp_lc_1            |    2365 | Nokia      | v20.6.0-13-gc6a313b84c |
+---------------------+---------+------------+------------------------+
```

3.3 Launching a container topology

This procedure launches a container topology using an automated script. See Figure 1 for a topology example.

Figure 1 **Example 3 node container topology**



Note: The docker-topo uses the SR Linux license at *~/docker-topo-master/topo-extra-files/examples/v2/config/license.key*.

Step 1. Copy the file *srlinux-vX.Y.Z-N.tar.xz* into ~/ of the Centos 7 Host Machine.

Step 2. Load the docker image:

```
$ docker image load -i ~/srlinux-vX.Y.Z-N.tar.xz
```

Example:

```
~ $ docker image load -i srlinux-vX.Y.Z.tar.xz
ba6c2307b523: Loading layer  329.1MB/329.1MB
53a1d27f1cbd: Loading layer   5.12kB/5.12kB
b1cfd1c17e46: Loading layer  311.3kB/311.3kB
559e14cb1ef4: Loading layer  303.1kB/303.1kB
99b5d14edc41: Loading layer  4.015MB/4.015MB
b89e0ee26076: Loading layer  3.072kB/3.072kB
ba4b6c40087a: Loading layer   2.56kB/2.56kB
4f529c55a3c9: Loading layer   2.56kB/2.56kB
9637194042f7: Loading layer   7.68kB/7.68kB
34eed33aab83: Loading layer   25.6kB/25.6kB
```

```
964b072de0ec: Loading layer   25.6kB/25.6kB
ac434023ca76: Loading layer  3.072kB/3.072kB
ae35e17f1206: Loading layer  3.072kB/3.072kB
7f230658f8b8: Loading layer  3.072kB/3.072kB
78f9091adeb6: Loading layer   2.56kB/2.56kB
b8f40a106975: Loading layer  3.072kB/3.072kB
238c0f5a7a42: Loading layer  3.072kB/3.072kB
c607a3d0a237: Loading layer  3.072kB/3.072kB
00f1c823da02: Loading layer  3.072kB/3.072kB
068f6071eab8: Loading layer  403.9MB/403.9MB
1963843423a6: Loading layer  3.072kB/3.072kB
Loaded image: srlinux:v<release>-190
```

Step 3. Check that the docker image was imported correctly:

```
$ docker images
```

Example:

```
REPOSITORY TAG          IMAGE ID              CREATED         SIZE
srlinux 19.11.1      3e77d45745f2          7 hours ago     1.3GB
```

Step 4. Create a *docker-topo-master* folder with the required files, using the following command:

```
# cd ~; tar -zxvf docker-topo-master.tar.gz
```

Step 5. Change directories to the docker-topo-master directory:

```
cd ~/docker-topo-master
```

Step 6. In the docker-topo-master directory, install docker-topo using:

```
# sudo python3.6 setup.py install
```

Note: If you encounter an error creating a directory or file, check the directory permissions.

Step 7. The image name `SRLINUX_IMAGE: srlinux:vX.Y.Z-N` is defined by default. If a different name is required, edit the image name in file *~/docker-topo-master/topo-extra-files/examples/v2/3-node.yml*.

Example (image filename location):

```
VERSION: 2
driver: veth
PREFIX: 3-node
SRLINUX_IMAGE: srlinux:19.11.1
PUBLISH_BASE: 9000
links:
  - endpoints: ["srlinux-A:e1-1", "srlinux-B:e1-1"]
  - endpoints: ["srlinux-A:e1-2", "srlinux-C:e1-2"]
  - endpoints: ["srlinux-B:e1-3", "srlinux-C:e1-3"]
```

Step 8. **Important:** If there are already existing containers with 3-node.yml, they need to be destroyed before creating the new 3-node topology. See Destroying an existing topology for this procedure.

If there is no existing topology (or an existing topology is destroyed), create a 3-node topology using:

```
$ docker-topo --create topo-extra-files/examples/v2/3-
node.yml
```

Example:

```
$ docker-topo --create topo-extra-files/examples/v2/3-node.yml
INFO:__main__:Version 2 requires sudo. Restarting script with sudo
INFO:__main__:
alias srlinux-A='ssh -o UserKnownHostsFile=/dev/null -
o StrictHostKeyChecking=no admin@$(docker inspect 3-node-srlinux-A --
format "{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}")'

$ alias srlinux-B='ssh -o UserKnownHostsFile=/dev/null -
o StrictHostKeyChecking=no admin@$(docker inspect 3-node-srlinux-B --
format "{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}")'

$ alias srlinux-C='ssh -o UserKnownHostsFile=/dev/null -
o StrictHostKeyChecking=no admin@$(docker inspect 3-node-srlinux-C --
format "{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}")'
INFO:__main__:All devices started successfully
```

Note: Topology for 1-Node and 2-Nodes can also be built using the following commands:

- For 2-Node: `$ docker-topo --create topo-extra-files/examples/v2/2-node.yml`
- For 1-Node: `$ docker-topo --create topo-extra-files/examples/v2/1-node.yml`

Step 9. Using the output from the previous step, create login shortcut commands.

Example:

```
$ alias srlinux-A='ssh -o UserKnownHostsFile=/dev/null
-o StrictHostKeyChecking=no admin@$(docker inspect 3-
node-srlinux-A --format "{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}")'


$alias srlinux-B='ssh -o UserKnownHostsFile=/dev/null -
o StrictHostKeyChecking=no admin@$(docker inspect 3-
node-srlinux-B --format "{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}")'
```

```
$ alias srlinux-C='ssh -o UserKnownHostsFile=/dev/null
-o StrictHostKeyChecking=no admin@$(docker inspect 3-
node-srlinux-C --format "{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}")'
```

Step 10. Once the shortcut is created, use the alias command to open the CLI with the alias name. Enter **admin** as the username and password. For example, to open srlinux-A CLI, enter:

```
$ srlinux-A
```

Example:

```
$ srlinux-A
Warning: Permanently added '172.18.18.2' (RSA) to the list of known hosts.
admin@172.18.18.2's password:
Hello admin,
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ running }--[  ]--
```

Step 11. Check that new nodes are up and running:

```
$ docker ps -a
```

Example:

```
$ docker ps -a
CONTAINER ID    IMAGE            COMMAND                 CREATED         STATUS       PORT       NAMES
2505ea85a702    srlinux:v0.1.0-198  " /tini -- fixuid -q …"2 mins ago   Up 2 minutes           3-
node-srlinux-A
f8a0b21f126d    srlinux:v0.1.0-198  " /tini -- fixuid -q …"2 mins ago   Up 2 minutes           3-
node-srlinux-B
b30f6326c87b    srlinux:v0.1.0-198  " /tini -- fixuid -q …"2 mins ago   Up 2 minutes           3-
node-srlinux-C
```

Step 12. Access the CLI for the nodes.

a. Using the shortcut command created in Step 9:

```
srlinux-A
```

- If the shortcut was not created, use the docker exec command:

```
$ docker exec -it 3-node-srlinux-A sr_cli
```

b. Using the full SSH command, as shown in the output after the container creation:

```
$ ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no admin@$(docker inspect 3-
node-srlinux-A --format "{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}")
```

Note: If the CLI prompt does not come up immediately, adjust the terminal window size. This is a known Docker issue.

Step 13. Verify that the interface and IP addresses exist on all nodes:

i. `enter running`

ii. `info interface <interface name>`

iii. `info network-instance <instance name>`

Example:

```
# enter running
--{ running }--[  ]--
# info interface *
    interface ethernet-1/1 {
        admin-state enable
        subinterface 1 {
            admin-state enable
            ipv4 {
                dhcp-client false
                address 192.168.11.1/30 {
                }
            }
        }
    }
    interface ethernet-1/2 {
        admin-state enable
        subinterface 1 {
            admin-state enable
            ipv4 {
                dhcp-client false
                address 192.168.12.1/30 {
                }
            }
        }
    }
    interface mgmt0 {
        admin-state enable
        subinterface 0 {
            admin-state enable
            ipv4 {
                dhcp-client true
            }
            ipv6 {
                dhcp-client true
            }
        }
    }
--{ running }--[  ]-
# info network-instance *
    network-instance mgmt {
        type ip-vrf
        admin-state enable
        description "Management network instance"
        ip-forwarding {
            receive-ipv4-check true
            receive-ipv6-check true
        }
        interface mgmt0.0 {
        }
        protocols {
            linux {
                export-routes true
                export-neighbors true
            }
        }
    }
    network-instance red {
        type ip-vrf
        admin-state enable
        ip-forwarding {
            receive-ipv4-check true
            receive-ipv6-check true
        }
        interface ethernet-1/1.1 {
        }
        interface ethernet-1/2.1 {
        }
        protocols {
            linux {
                export-routes true
                export-neighbors true
            }
        }
        static-routes {
            route 192.168.13.0/24 {
                admin-state enable
                metric 1
                preference 5
                next-hop-group static-ipv4-grp
            }
        }
        next-hop-groups {
            ecmp {
                max-paths-level-2 1
            }
            group static-ipv4-grp {
                admin-state enable
                collect-stats false
                nexthop 1 {
                    ip-address 192.168.12.2
                    admin-state enable
                    resolve true
                }
                nexthop 2 {
                    ip-address 192.168.11.2
                    admin-state enable
                    resolve true
                }
            }
        }
    }
--{ running }--[  ]--
```

Step 14. Ping different nodes to verify connectivity:

```
ping <IP Address> network-instance <instance name>
```

Example:

```
--{ running }--[  ]--
# ping 192.168.11.1 network-instance red
Pinging 192.168.11.1 in srbase-red
PING 10.0.0.0 (192.168.11.1) 56(84) bytes of data.
64 bytes from 192.168.11.1: icmp_seq=1 ttl=64 time=1.74 ms
64 bytes from 192.168.11.1: icmp_seq=2 ttl=64 time=0.596 ms
64 bytes from 192.168.11.1: icmp_seq=3 ttl=64 time=0.664 ms
64 bytes from 192.168.11.1: icmp_seq=4 ttl=64 time=0.710 ms
^C
--- 192.168.11.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.596/0.927/1.740/0.471 ms
```

3.4 Destroying an existing topology

Destroy an existing topology using the **docker-topo --destroy** command. Destroy the created topology using the following command:

```
$ docker-topo --destroy topo-extra-files/examples/v2/3-
node.yml
```

4 Installing software

Software installation tasks include:

- Installation concepts
  Describes concepts to be familiar with prior to installing or upgrading.
- Installing the software manually on a 7250 IXR
  Installs the SR Linux software on a 7250 IXR system for the first time.
- Installing the software manually on a 7220 IXR-D
  Installs the SR Linux software on a 7220 IXR-D system for the first time.
- Upgrading the software
  Upgrades previously installed SR Linux software to a higher version.

4.1 Installation concepts

SR Linux can be installed on either the 7250 IXR or 7220 IXR-D 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.

4.1.1 7250 IXR installation concepts

On a 7250 IXR system, SR Linux boots from an internal SD card. No other boot device may be used with the system. The SD card 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).

4.1.2 7220 IXR-D installation concepts

On a 7220 IXR-D system, SR Linux boots from a the internal SSD. No other boot device may be used with the system. The internal SSD contains:

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

SR Linux can boot from a USB device in recovery scenarios.

4.1.3 Software image

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 gzipped tarball containing these files, along with some other files used for operations and maintenance (for example, YANG models and SNMP MIBs).

4.2 Installing the software manually on a 7250 IXR

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

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.

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 command:

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

```
# cp <path-to-srlinux-rescue-image-tgz>
<destinationdirectory>
```

Example:

```
# cp /mnt/removable/SRLinux-20.6.1-10.tgz /tmp
# cp /mnt/removable/initramfs_rescue-4.19.39-2.x86_64-03.img /tmp
```

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

Step 3. Install x86_64-efi on the system.

Example:

```
# sudo yum install grub2-efi-x64-modules
```

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

Example:

```
# wget https://rpmfind.net/linux/fedora/linux/releases/29/Everything/x86_64/os/
Packages/d/dosfstools-4.1-6.fc29.x86_64.rpm
# sudo yum localinstall dosfstools-4.1-6.fc29.x86_64.rpm
```

Step 5. Download the sdcardflash.sh script.

Step 6. Run the script.

Example:

```
# /tmp/sdcardflash.sh  -v -e 20.6.1-10 -i srlinux-20.6.1-10.tgz -r initramfs_rescue-
```

```
4.19.39-2.x86_64-03.img -s /dev/sdb -g "autoboot nosinstall"
```

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 module (if applicable).

Step 9. Remove both control modules from the system (refer to the *SR Linux 7250 Hardware Installation Guide* for a procedure), then insert the SD cards into the internal SD slot for each module.

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

4.3 Installing the software manually on a 7220 IXR-D

Installing the software manually on a 7220 IXR-D system requires a working Linux system and a USB device. Installation also requires the ONIE boot loader install environment.

If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrap procedure to complete the installation.

Step 1. Download the ONIE recovery .iso image for the respective IXR-D unit 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. Once 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 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: OACHI037 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**.

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

Step 7. Install the ONIE from the USB. Select **ONIE: Embed ONIE** in the GNU Grub screen.

Step 8. Once the ONIE installation is complete, remove the USB to boot ONIE from the SSD.

Step 9. Once the device boots 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 begins to get 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 Manual bootstrapping procedure to continue.

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

4.3.1 Manual bootstrapping

If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrapping to retrieve the image. Perform this procedure in addition to the steps in Installing the software manually on a 7220 IXR-D to complete the installation.

Step 1. After the ONIE image boots the service discovery starts automatically. To stop the service discovery, execute:

```
ONIE:/ # onie-stop
```

Step 2. Configure the management IP address and the default route to copy the SR Linux image to the 7220 IXR-D.

```
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 3. Using the `SCP` command, copy the SR Linux image <version>.bin to the root folder. The "root" user password is blank.

Step 4. 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 5. Once the image is installed, the 7220 IXR-D reboots with the SR Linux image.

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

SRLINUX 20.6.1-21463
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 6. Enter the login credentials: `linuxadmin`

Step 7. Disable watchdog reboot using the following command:

```
sr_wdc noreboot
```

Step 8. Permanently disable the ZTP service using the following command.

```
systemctl disable ztp
```

Step 9. Enable SR Linux as a service with the `systemctl` command.

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

Step 10. Configure the network IP address and enable network as service by performing the following steps.

i. Enter `sudo bash -c 'echo "NETWORKING=yes" >/etc/sysconfig/network'`

ii. Enter `sudo systemctl enable network`

iii. Edit and include the `sudo/etc/sysconfig/network-scripts/ifcfg-mgmt0` with the appropriate IP address, netmask, and gateway information.

```
DEVICE=mgmt0
IPADDR=<IP_ADDR>
BOOTPROTO=static
NETMASK=<NET_MASK>
GATEWAY=<GATEWAY>
ONBOOT=yes
IPV6INIT=NO
NM_CONTROLLED=no
```

Step 11. After a reboot, the networking and SR Linux service is started automatically.

4.4 Upgrading the software

This procedure upgrades the software using the CLI. It requires a working system, with SR Linux operational and the CLI available. If the system is not operational and CLI is not available, see either Installing the software manually on a 7250 IXR or Installing the software manually on a 7220 IXR-D to perform an initial installation.

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-20.6.1-10.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: `linuxadmin`

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

a. 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-20.6.1-10.bin /local-destination
```

b. 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 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/20.6.1-10
# sudo cp /mnt/removable/srlinux-20.6.1-10.bin /tmp/srlinux-20.6.1-10.bin
# sudo chmod +x /tmp/srlinux-20.6.1-10.bin
# sudo /tmp/srlinux-20.6.1-10.bin --target /mnt/nokiaos/20.6.1-10 --noexec
```

Step 6. Start an SR Linux CLI session, and retrieve the current version of the software:

Example:

```
# sr_cli
# info from state system boot image
    system {
         boot {
            image [
               19.11.7-16
                  ]
               }
           }
```

Step 7. Update the boot image list by reordering the current version behind the new version:

```
# enter candidate
# system boot image [ <version> <old-version> ]
# commit now
```

Example:

```
# enter candidate
# system boot image [ 20.6.1-10 19.11.7-16 ]
# commit now
```

Step 8. Reboot the chassis:

```
# tools platform chassis reboot
```

Step 9. Wait up to ten minutes, then log in to the device via SSH or console, and confirm the new version.

5 Zero Touch Provisioning

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.

Zero Touch Provisioning (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.

5.1 Applicability

The following implementation is currently supported:

- auto-boot using 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)

5.2 Overview

For auto-boot using OOB, the node storage device ships with the SR Linux image and the grub.cfg. Within the grub.cfg is an auto-boot flag. The correct part number should be ordered to obtain the grub.cfg with the auto-boot flag enabled by default. The flag can be manually changed if needed.

When the SR Linux boots, it checks the grub.cfg for the auto-boot flag. If the flag is set, the node goes into the auto-boot mode.

Once initiated, the auto-boot mode starts the auto-provisioning process. The 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 provision 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. Once 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 will not enter auto-boot mode.

5.2.1 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 2 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 2 **All components in the same subnet**

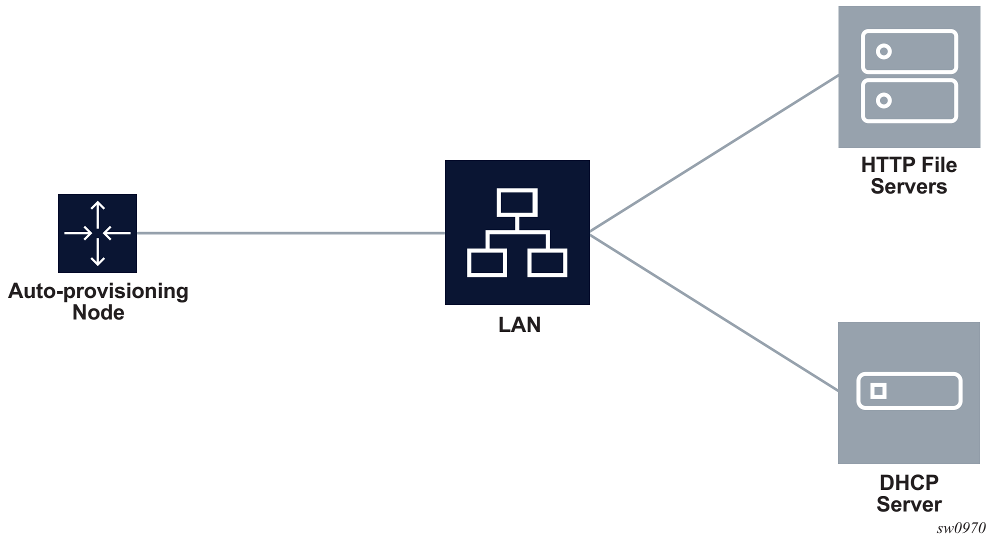


Figure 3 shows the second scenario where only the HTTP file servers and DHCP server are in the same subnet. The DHCP relay is used to fill Option 82 as the gateway address. The gateway address is used to find the appropriate pool in the DHCP server to assign the correct subnet IP address to the SR Linux.

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 3 **HTTP file and DHCP servers in the same subnet**

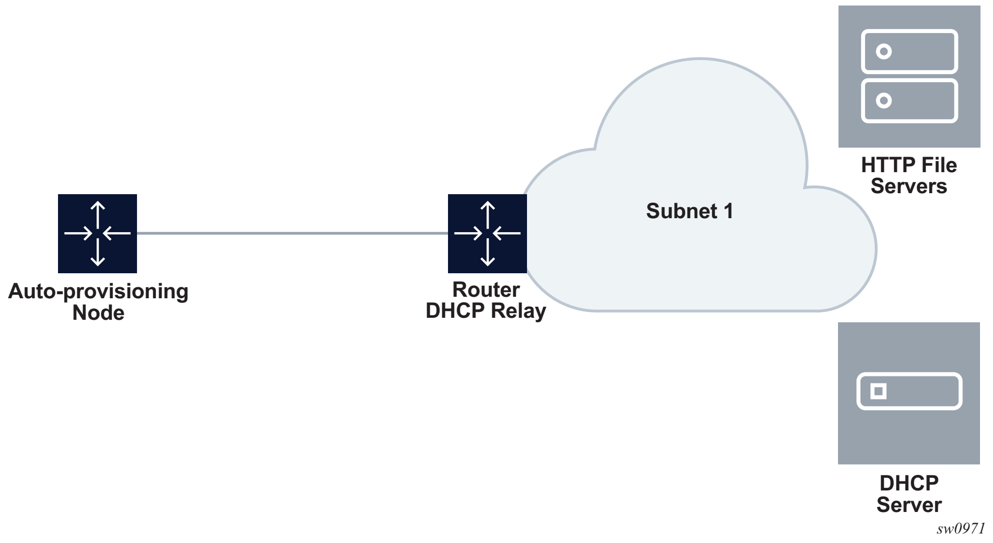
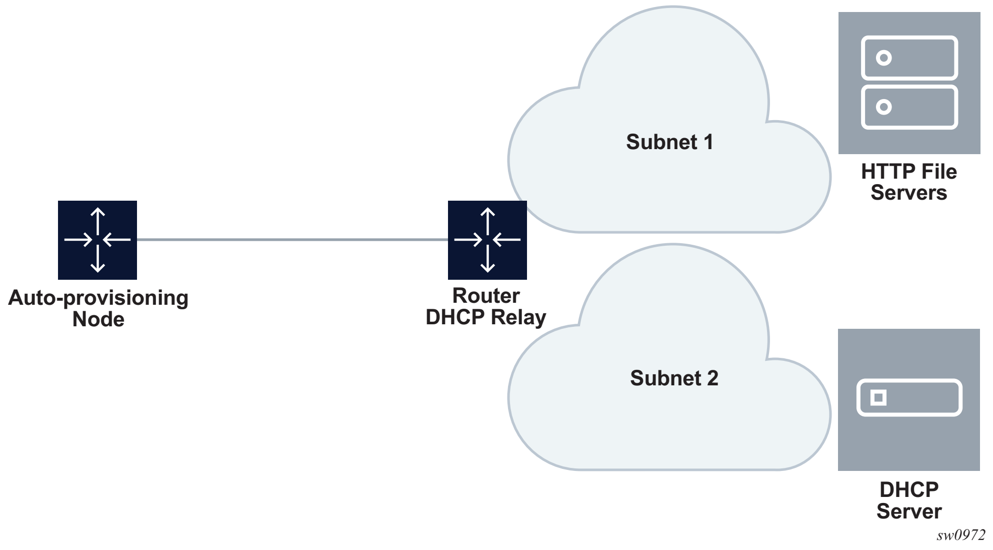


Figure 4 shows the third scenario where all components are in different subnets. The DHCP relay adds the Option 82 gateway address to the DHCP request, and the DHCP server adds the Option 3 with the gateway address of the HTTP file server.

Figure 4 **All components in different subnets**



5.3 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.

5.3.1 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 will be 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 will be 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 will assign the IP address to the node. The DHCP offer for IPv4 should contain the options shown in Table 4.

Table 4 **Required DHCP offer options**

Option	Name	Description
viaddr	Client-Ip-Address	Network interface – IP address (for NW 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 host name	Server IP address
67	Bootfile Name	URL or IP to the provisioning file

Table 5 defines DHCP IPv4 and IPv6 equivalents.

Table 5 **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	—
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	—

5.3.1.1 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 will perform 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 References for procedures and commands used to provision available options.

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

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

5.3.2 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, like 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.

5.3.2.1 Default gateway route configuration for IPv4

Once 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).

5.3.2.2 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.

5.3.3 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 will be 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 and places it on the storage device. The node then uses the Python provisioning script to download any RPMs, images, scripts, or config and places them in 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 once 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 will not enter auto-boot mode. Once the nodes reboot, they will come up in an operational state with the config and image.

For additional information about the Python provisioning script, see Configuring the Python provisioning script.

5.3.4 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 config failed (possibly due to the server not being available, or incorrect directory or credentials).
- Copying the RPMs, scripts, or config 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.

5.3.5 ZTP log files

ZTP log files are stored under */var/log/ztp*. For 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]#
```

5.4 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 the SR Linux CLI
- Checking the status of a ZTP process using the SR Linux CLI

5.4.1 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.

Refer to the References sections for a complete list of available ZTP-related commands.

5.4.2 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 config
- 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

The following is an example Python provisioning script.

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()
```

5.4.3 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 5.4.4 Configuring options in the grub.cfg using ZTP CLI)
- SR Linux CLI (see procedure 5.4.10 Configuring options in the grub.cfg using SR Linux CLI)

5.4.4 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:

Command	Description
autoboot	Enables or disables the auto-boot flag
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 if 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 describes examples of **ztp option** commands and available arguments.

Table 6 **ZTP CLI: ztp option command examples**

Command and description	Command syntax
autoboot Enable or disable the auto-boot flag	`ztp option autoboot --status [enable | disable]`
bootintf Specify the specific boot interface to send DHCP over	`ztp option bootintf --intf` *`<name of interface>`*
bootintf Remove the previously set boot interface	`ztp option bootintf --remove`

duration
Set the ZTP timeout value
	`ztp option duration --timeout` *`<integer in seconds>`*
	Default=3600, Range=200-3600

duration
Set the number of ZTP retry attempts
	`ztp option duration --retry` *`<integer>`*
	Default=3, range=1-10

clientid Set the client ID to a chassis MAC address	`ztp option clientid --type mac`
clientid Set the client ID to a serialID	`ztp option clientid --type serialid`
nosinstall Enable or disable NOS upgrade flag	`ztp option nosinstall --status [enable | disable]`
list Display the current value of each option	`ztp option list`

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     |
+--------------+----------+
```

5.4.5 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:

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 7 describes examples of **ztp image** commands and available arguments.

Table 7 **ZTP CLI: ztp image command examples**

Command and description	Command syntax
Activate Activate a specific NOS image	`ztp image activate --version <build version> [--no-reboot]` **Note**: `--no-reboot` means do not reboot after activate to take new build into use.

Table 7 **ZTP CLI: ztp image command examples (Continued)**

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

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   |
+-------------+---------+
```

5.4.6 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>
```

5.4.7 Redownloading the executable files with ZTP CLI

Executable files (RPMs, scripts, and configs) can be redownloaded if required. This 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>
```

5.4.8 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:

Command	Description
canstart	Indicates whether ZTP can be started in current condition
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 8 describes examples of **ztp service** commands and available arguments.

Table 8 **ZTP CLI: ztp service command examples**

Command and description	Command syntax
canstart Verify whether ZTP can start in current condition	`ztp service canstart`
start Start the ZTP process	`ztp service start`
start Start the ZTP process and enable/disable auto-boot	`ztp service start --autoboot [enable | disable]`
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`

Table 8 **ZTP CLI: ztp service command examples (Continued)**

Command and description	Command syntax
restart Restart the ZTP process and enable/ disable auto-boot	`ztp service restart --autoboot [enable | disable]`

5.4.9 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
```

Output Example:

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

5.4.10 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:

Command	Description
admin-state	Enables or disables the auto-boot functionality
interface	Sets the interface used for the auto-boot functionality

Command	Description
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 9 describes examples of SR Linux commands and available arguments.

Table 9 **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 [enable	disable]` Default=enable

interface
Specify the specific boot interface to send DHCP over
	`system boot autoboot interface <`*`name of interface`*`>`
	Default=mgmt0

timeout
Set the ZTP timeout value
	`system boot autoboot timeout <`*`integer in seconds`*`>`
	Default=3600, Range=200-3600

attempts
Set the number of ZTP retry attempts
	`system boot autoboot attempts <`*`integer`*`>`
	Default=3, range=1-10

client-id Set the client ID to a serial ID or a chassis MAC address	`system boot autoboot client-id [serial	mac]` Default=serial

5.4.11 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:

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

where *command* must be the following:

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

Table 10 describes an example of the SR Linux command and available argument.

Table 10 **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	`system boot image <ordered list of images>` **Note**: Up to 3 files can be specified.

5.4.12 Starting, stopping, and restarting a ZTP process using the 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:

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 11 describes examples of SR Linux commands and available arguments.

Table 11 **SR Linux CLI: start, stop, and restart process command examples**

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

5.4.13 Checking the status of a ZTP process using the 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
```

5.5 References

5.5.1 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
        — canstart
            — [--format table | json]
        — restart
            — --autoboot enable | disable
        — start
            — --autoboot enable | disable
        — status
            — [--format table | json]
        — stop
            — --autoboot enable | disable
```

5.5.2 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 additional 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 ZTPClient. Each use of the ZTPClient passes through a call to one of its methods.

The path to the API client class:

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

Example:

```
import ztpclient
class ZTP(object):

    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()
    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
    if __name__ == '__main__':
        ztp = ZTP()
        ztp.perform_ztp()
```

Functions

chassis_control()

Lists control card information.

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	`>>> client.chassis_control()`
	`{u'status': 0, u'message': {u'operation': u'active'}}`
	`>>> client.chassis_control()`
	`{u'status': 0, u'message': {u'operation': u'standby'}}`

chassis_linecards()

Lists line card information of the chassis.

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	`>>> client.chassis_linecards()`
	`{u'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}]}`

configure(configurl)

Downloads the configuration from a specific `configurl` and applies the configuration to SR Linux. If SR Linux service(s) is not running, the service(s) are started and the configuration is applied.

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`, then no action is performed. If there is no image in the chassis of specific `version`, then no action is performed. If the specified `version` is available, then chassis will be rebooted to that `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	`>>> client.image_list()`
	`{u'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740', u'20.6.1-17738']}`
	`>>> client.image_activate('20.6.1-3333')`
	`{u'status': 127, u'message': u'20.6.1-3333 is not available'}`
	`>>> client.image_activate('20.6.1-18836')`
	`{u'status': 127, u'message': u'20.6.1-18836 is current active version. No additional change required'}`

image_bootorder(bootorder)

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

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 valuse otherwise.

Information	Description

Examples	`>>> client.image_bootorder(['20.6.1-18836','20.6.1-17740','20.6.1-17738'])`
	`{u'status': 0, u'message': None}`
	`>>> client.image_bootorder('20.6.1-18836,20.6.1-17740,20.6.1-17738')`
	`{u'status': 0, u'message': None}`

image_delete(version)

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

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	`>>> client.image_list()`
	`{u'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')`
	`{u'status': 0, u'message': u'20.6.1-3333 version not available'}`
	`>>> client.image_delete('20.6.1-18836')`
	`{u'status': 127, u'message': u'Cannot remove active version'}`

image_list()

Lists all currently available image versions on the hardware.

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 the list of images. The item in list `*` indicates the current active image version. **Note**: The `image_list` does not indicate the boot order.

Examples	`>>> client.image_list()`
	`{u'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740', u'20.6.1-17738']}`

image_upgrade(image_url, md5_url, options)

Performs an image upgrade.

Information	Description
Arguments	*image_url* (string): The URLfrom where the image should be downloaded. *md5_url* (string): The URL from where the pre-calculated md5sum of the image should be downloaded. Once image is downloaded, the calculated md5sum is checked against the downloaded md5sum. If the values do not match, then image upgrade is discarded. *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. *skip_check* (boolean): If set to true, skip status of `autoboot` parameter 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. *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. **Note**: Based on the setting and outcome, the chassis can be rebooted when invoking this API. **Note**: To perform ZTP, the `autoboot` flag must be enabled
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	—

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.

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	`>>> client.option_autoboot(ztpclient.ZtpStatus.enable)`
	`{u'status': 0, u'message': None}`

option_bootintf(interface)

Sets the interface to be used by ZTP in various procedures. Default value is `mgmt0`.

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	`>>> client.option_bootintf('mgmt0')`
	`{u'status': 0, u'message': None}`

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.

Information	Description
Arguments	*type* (ZtpClientId): The client identifier type.
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	`>>> client.option_clientid(ztpclient.ZtpClientId.serialid)`
	`{u'status': 0, u'message': None}`
	`>>> client.option_clientid(ztpclient.ZtpClientId.mac)`
	`{u'status': 0, u'message': None}`

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 higher version image to lower version). When the option is disabled, only upgrades are allowed.

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.

Information	**Description**

Examples	`>>> client.option_downgrade(ztpclient.ZtpStatus.enable)`
	`{u'status': 0, u'message': None}`

option_duration(timeout, retry)

Sets the `timeout` and `retry` parameters of the ZTP process. If not successful, the ZTP process keeps re-trying for the specified `timeout` seconds. Once the timeout is reached, the process stops. If the number of attempts are equal to the `retry` value, then the specified action is taken. The default action is to reboot.

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	`>>> client.option_duration(3600,3)`
	`{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 enabled, the NOKIA-DATA overlay file system is formatted. Any change performed on the overlay file system will be removed.

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	`>>> client.option_formatovl(ztpclient.ZtpStatus.enable)`
	`{u'status': 0, u'message': None}`

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 enabled, the NOKIA-ETC overlay file system is formatted. Any change performed on the overlay file system will be removed.

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	`>>> client.option_formatsrletc(ztpclient.ZtpStatus.enable)`
	`{u'status': 0, u'message': None}`

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 enabled, the NOKIA-OPT overlay file system is formatted. Any change performed on the overlay file system will be removed.

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	`>>> client.option_formatsrlopt(ztpclient.ZtpStatus.enable)`
	`{u'status': 0, u'message': None}`

option_list()

Lists all the options of the ZTP process.

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.

Information	Description
Examples	`>>> 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}}`

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.

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	`>>> client.option_nosinstall(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

provision(provisionurl)

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

Information	Description
Arguments	*provisionurl* (string): The URL from where the provisioning script 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. **Note**: If the script returns a non-zero exit code, then 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. **Note**: To perform ZTP, the `autoboot` flag must be enabled
Examples	`>>> client.provision('http://135.227.248.118/duts/IDNS1833F0766/srlinux_ztp.py')`

service_restart()

Restarts the ZTP service.

Information	**Description**
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in `message` attribute. `status` attribute is set to 0 if successful, non-zero otherwise.

Examples	`>>> client.service_restart()`
	`{u'status': 0, u'message': {u'status': u'Service started'}}`

service_start()

Starts the ZTP service (if not already running).

Information	**Description**
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in `message` attribute. The `status` attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.service_start()`
	`{u'status': 0, u'message': {u'status': u'Service started'}}`

service_status()

Gets the current status of the ZTP service. The ZTP service will be running as systemd service. It can be checked manually by running ‘systemctl status ztp’.

Information	**Description**
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in `message` attribute. The `status` attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.service_status()`
	`{u'status': 0, u'message': {u'status': u'Inactive'}}`
	`>>> client.service_status()`
	`{u'status': 0, u'message': {u'status': u'Active'}}`

service_stop()

Stops the ZTP service (if already running).

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in `message` attribute. The `status` attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.service_stop()`
	`{u'status': 0, u'message': {u'status': u'Service stopped'}}`

Customer Document and Product Support

Customer Documentation

Customer Documentation Welcome Page

Technical Support

Product Support Portal

Documentation Feedback

Customer Documentation Feedback